# Exascale Conference: Performance Modeling and Metrics Discussion

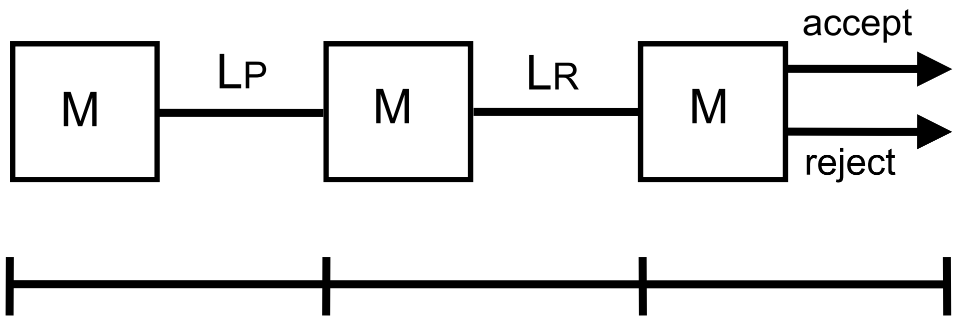
K. J. Roche
High Performance Computing Group, Pacific Northwest National Laboratory

October 2, 2012
Arlington, Virginia

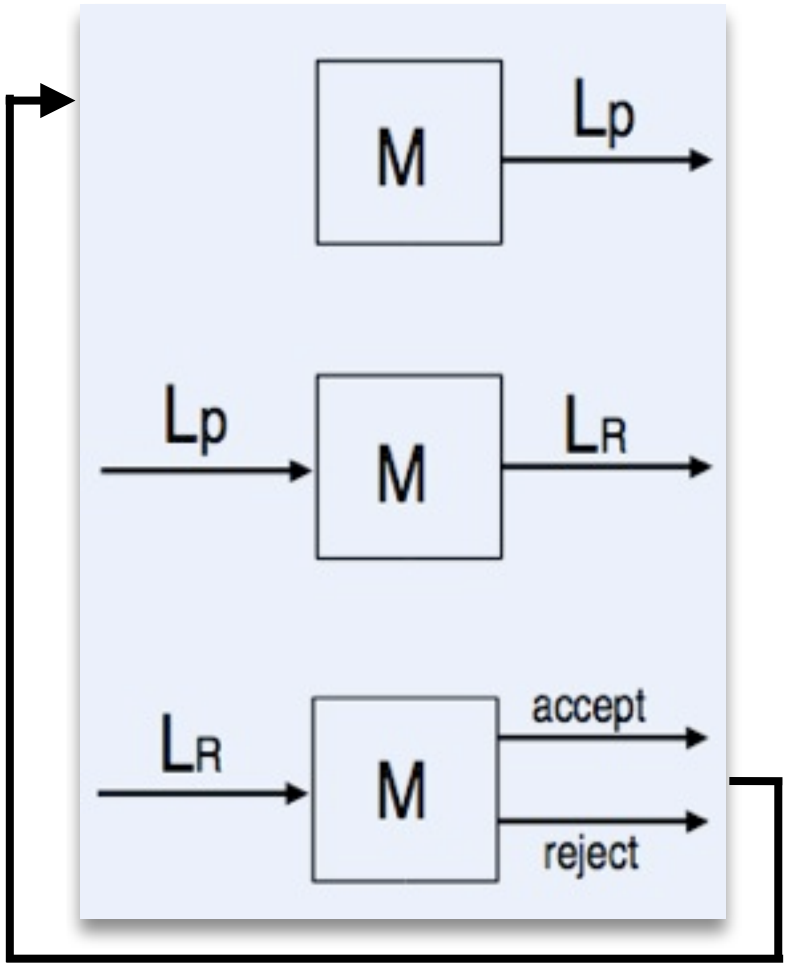
- this talk is intended to start conversations and describe the kinds of issues encountered in DOE’s exascale performance modeling and simulation work   -- mostly observations, not a research talk

# Concepts: Solving Problems with Computers

- COMPLEXITY
  - PROBLEMS
  - ALGORITHMS
  - MACHINES



Measured time for machine M to generate the language of the problem plus time to generate the language of the result plus the time to accept or reject the language of the result.



Asking questions, solving problems is recursive process

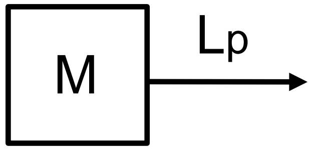
Accepting a result means a related set of conditions is satisfied

S = S1 ^ S2 ^ ... ^ Sn

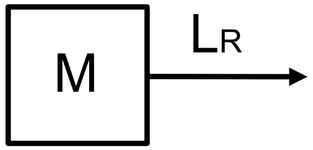
***algorithm***, a Turing machine that always halts
***decidable problems*** are posed as a recursive language
***undecidable problems*** have no algorithms that accept the language of the problem and generate / accept or reject an answer (Rice's Theorem posits that non-trivial properties of r.e. languages are undecidable. Examples are emptiness, finiteness, regularity, and context freedom.)

# Metrics (?) to Judge Platforms; Extended Scope of Application Software

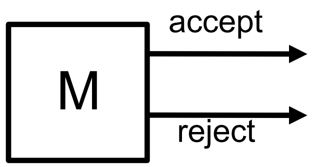
Example Problem: solving algebraically determined systems of linear equations numerically (Linpack TOP500, **FLOPs**)



• construct A, b given dim(A) dim(A)=dim(rank(1,A))=dim(rank(2,A))

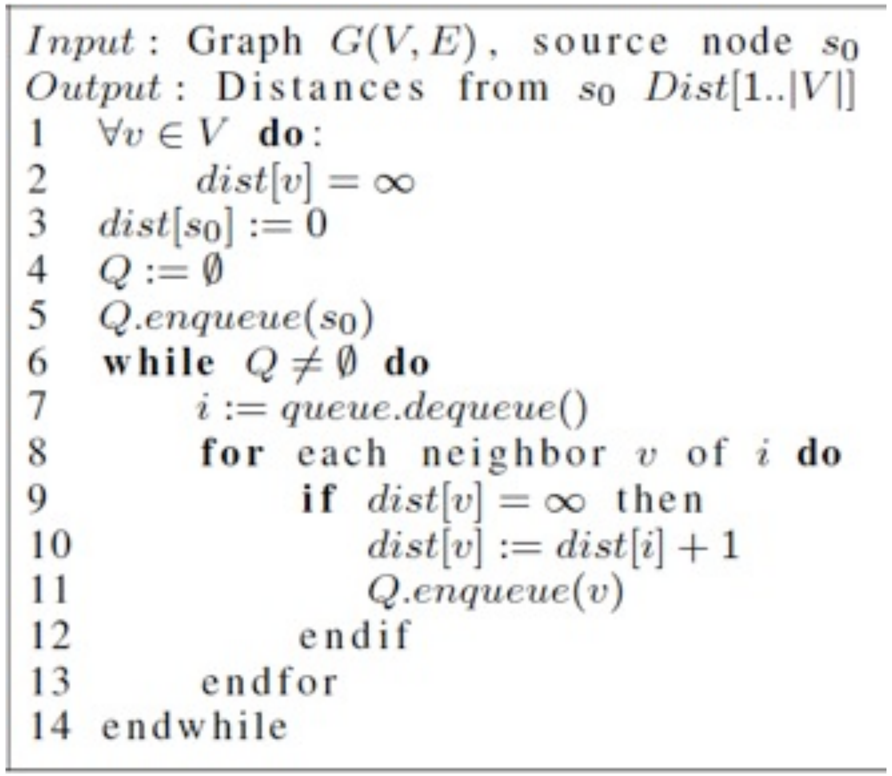


• compute A=LU, solve Ly=b for y, solve Ux=y for x



• compute $|A|_{inf}$ , compute $|x|_2$ , compute $|b-Ax|_2$ , approximate $\varepsilon$ (machine precision), form $\kappa= |b-Ax|_2 / (|A|_{inf} * |x|_2 * \varepsilon)$ ; if $\kappa \sim$ O(dim(A)) **accept** x, else **reject** x

Ex2: BFS(Graph500,**TEPS**)

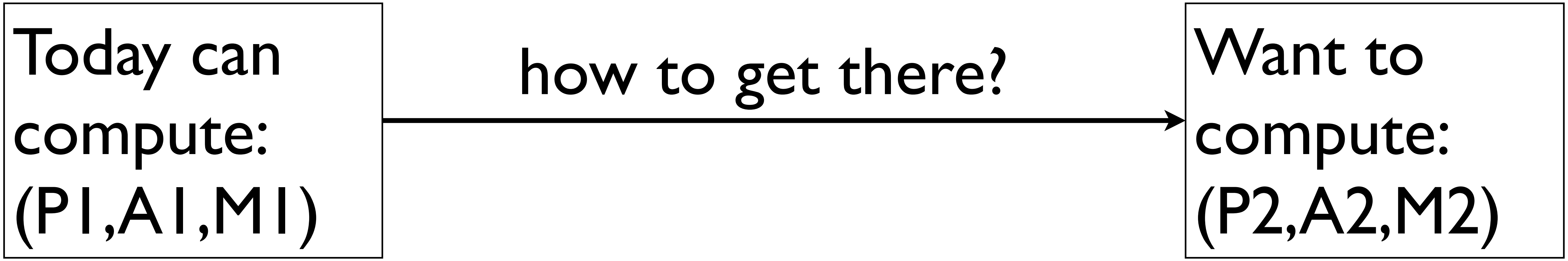


```
Input : Graph $G(V,E)$, source node $s_0$
Output : Distances from $s_0$ $Dist[1..|V|]$
$\forall v \in V$ do :
        $dist[v] = \infty$
$dist[s_0] := 0$
$Q := \emptyset$
$Q.enqueue(s_0)$
while $Q \neq \emptyset$ do
        $i := queue.dequeue()$
        for each neighbor $v$ of $i$ do
                if $dist[v] = \infty$ then
                        $dist[v] := dist[i] + 1$
                        $Q.enqueue(v)$
                endif
        endfor
endwhile
```

Q: How do the language of the problem and the accepted result relate to reality?
Requires analysis beyond software analysis above and distinguishes ***computational science*** from system and library software development. Takes more time -needs refinement phase of **algorithms and metrics**.

**Metric:** the distance between two points in some topological space

# Top Down Perspective : Application scientists / developers



Today can compute: (P1,A1,M1)

how to get there?

Want to compute: (P2,A2,M2)

limits:

- don’t really understand the problem, $\boxed{M} \xrightarrow{L_p}$ ??
- don’t have the right method / representation (A2: $\boxed{M} \xrightarrow{L_R}$ ??)
- capacity (|M| too simple) -too much complexity (|P2| >> |P1|)
- takes too long (T(P,A,M) big)
    - fix it through software engineering and optimization
    - expose parallelism to take advantage of existing computers
- need carefully defined measures of scientific progress, $\boxed{M}\begin{array}{l}\xrightarrow{\text{accept}}\\ \xrightarrow[\text{reject}]{}\end{array}$

# Bottom Up Perspective : How to Build a BIG, PROGRAMMABLE Computer w/in a Constrained Power Envelope

what is the design of M?
what is the interface to M?

... too many issues to address in 10 minutes

# Clear need for both generic and specific metrics / models from both perspectives; also to connect the perspectives

Perpectives are different but events directly related

- hardware cannot generalize instruction and data miss rates from application to application

- applications cannot safely generalize program execution behavior from one architecture to another

SO, metrics are measures that help us reason about where things stand, how changes impact our goals

- models allow us to reason about and identify new metrics for existing infrastructure

- models also allow us to reason about and contemplate new metrics in the absence of infrastructure, i.e. design

# ASCR’s Annual OMB Software Metric from FY04-FY11

·Game: satisfy the following efficiency measure

> *(SC GG 3.1/2.5.2) Improve computational science capabilities, defined as the average annual percentage increase in the* **computational effectiveness (either by simulating the same problem in less time or simulating a larger problem in the same time)** *of a subset of application codes. Efficiency measure: X%*

·vertical interrogation of DOE’s computing investments; generate complete *before and after* versions of cutting-edge production science software on cutting-edge computing platforms with reproducible, quantifiable enhancements

·position important (ie ASCAC selection process, usually from SciDAC or INCITE) software applications to solve complex problems on the most massively parallel open computing systems today

·over 33 major software applications enhanced, i.e. 4 went on to win or be finalists in G. Bell prize, a couple of *Nature* or *Science* articles ensued; over 200M CPU Hours for correction + problem execution analysis

# ASCR OMB Software Benchmark Trends (FY04 - FY11)

| | |
|---|---|
| climate research | 4 |
| condensed matter | 4 |
| fusion | 5 |
| high energy physics | 3 |
| nuclear | 2 |
| subsurface modeling | 2 |
| astrophysics | 2 |
| combustion chemistry | 4 |
| bioinformatics | 1 |
| math, data analytics | 2 |
| molecular dynamics, electronic structure | 3 |
| nuclear energy | 1 |
| Total | 33 |

ref. my DOE ASCAC talks
or contact me / DOE ASCR HQ for more info

- *the effort has made quantifiable advances to DOE ASCR’s missions*
  - state of the art approaches -> effective use of LCF resources
  - determine near term feasibility of application studies

- *focal points*
  - new / improve problem capabilities, efficiency, weak and strong scaling performance enhancements
  - determine new metrics, evaluate measuring techniques and tools
  - identify deficiencies and provide solutions

- *broad range of activities w/ over 200 collaborators*
  - sharpen / validate tools through use in production science scenarios
  - pinpoint compiler related uses / misuses
  - data formats and compositions in memory, in network, in file system
  - i/o algorithms for massive data sets, small data sets in massively parallel envs
  - evaluate algorithms / numerics / physical representations
  - architectural adjustments

- *demonstrate leadership for application developers / programmers*

****see November 2011 ASCAC talk for specific examples, or any of the annual reports on record at ASCR.**

# Measurements

- application specific measures / metrics (see bonus for examples)

- machine events

  -clear dependence on tools / hardware support to monitor hardware components activated during program execution

  --cycle count, disk accesses, floating point operation counts, instructions issued and retired, L2 data cache misses, maximum memory set size, number of loads / stores etc.

  - derived measures

    -efficiency, cycles per instruction (CPI) or floating point operations retired per second (FLOPs)

    -computational costs, CPU Hours (relates execution time to processing elements), etc.

- **USE TOOLS** to pinpoint insufficient parallelism, lock contention, and parallel overheads in threading and synchronization strategies

# Enhancement Modes

- *performance* (improve efficiency, scalability - weak or strong)
  - -data structures / discretizations, algorithms, libraries, language enhancements, compilers
- *scientific* (better accuracy, improved predictive power)
  - -physical models, the problem representation, validity of inputs, and correctness of computed results

**Strong Scaling**

| Machine Events | Q2 | Q4 |
|---|---|---|
| INS | 2.147E+15 | 2.1130E+15 |
| FP_OP | 5.896E+14 | 5.8947E+14 |
| PEs | 5632 | 11264 |
| Time[s] | 121.252233 | 57.222988 |

**INS:**
2113046508030116 /
2146627269408190 = **.9843**

**FP_OP:**
589469277576687 /
589624961638025 = **.9997**

**PEs:** 11264 / 5632 = **2**

**Time[s]:**
57.222988 / 121.252233 = **.472**

**Weak Scaling**

| Machine Events | Q2 | Q4 |
|---|---|---|
| INS | 5.18E+17 | 1.93E+18 |
| FP_OP | 4.63E+17 | 1.81E+18 |
| PEs | 7808 | 31232 |
| Time[s] | 25339 | 23791 |

**INS: 3.72**

**FP_OP: 3.92**

**PEs: 4**

**Time[s]: .938**

NB: k= T(Q4)*PEs(Q4)/
T(Q2)*PEs(Q2) ~ 3.756

**Improve Efficiency**

| Machine Events | Q2 | Q4 |
|---|---|---|
| INS | 3.16E+12 | 4.37E+11 |
| FP_OP | 5.50E+11 | 5.53E+11 |
| PEs | 1 | 1 |
| L2DCM | 823458808 | 34722900 |
| Time[s] | 826.494142 | 79.414198 |

**INS: 0.1381** (7.239x)

**FP_OP: 1.0053** (0.99475x)

**PEs: 1**

**L2DCM: 0.0422** (23.715x)

**Time[s]: 0.0961** (10.407x)

## “simulating the same problem in less time”

Algorithm, machine strong scaling :

Q4 problem := Q2 problem  
Q4 algorithm := Q2 algorithm  
Q4 machine ~ k * Q2 machine  
Q4 time ~ 1/k * Q2 time

Algorithm enhancements, performance optimizations:

Q4 problem := Q2 problem  
Q4 algorithm ~ enhanced Q2 algorithm  
Q4 machine := Q2 machine  
Q4 time ~ 1/k * Q2 time

*Could consider other variations: algorithm and machine are varied to achieve reduction of compute time

## “simulating a larger problem in same time”

Algorithm, machine weak scaling (100%):

Q4 problem ~ k * Q2 problem  
Q4 algorithm := Q2 algorithm  
Q4 machine ~ k * Q2 machine  
Q4 time := Q2 time

Algorithm enhancements, performance optimizations:

Q4 problem ~ k * Q2 problem  
Q4 algorithm ~ enhanced Q2 algorithm  
Q4 machine := Q2 machine  
Q4 time := Q2 time

*Could consider other variations: problem, algorithm and the machine are varied to achieve fixed time assertion

# Computational Efficiency

- Total elapsed time to execute a problem instance with a specific software instance (algorithm) on a machine instance

- Parallel
  - e(n,p) := Tseq (n) / ( p * T(n,p) )

**weighted:**

(t*nPEs/DOF)_b/(t*nPEs/DOF)_e

## From the QCD community : test sm, predict ckm me, qgp thermal studies

**Metrics:**

1. Efficiency of Dirac inverter (multiply a complex 3d matrix w/ a 3d vector w/accumulation)
2. Computation of fermion force term (s)
3. Evolution of trajectories w/ MC (s)
4. Dollars per flop

**Science cases:**

•3 Green's functions: Wilson, domain wall, Clover (Asqtad in MILC)

**Tunable parameters:**

•Size of the sub-lattice, V_0

•Number of processors, N_p (N_p := lattice size / V_0)

- *VisIt*, image construction / display time
- *RAPTOR*, time / cell / time step
- *XGC1*, time / time step / particle
- *DCA++*, time / disorder configuration
- *PFLOTRAN*, (time / DOF) / PE
- *GTC-s*, # particles / ( time / time step)
- *CHIMERA*, time / subcycled hydro step
- *S3D*, time / DOF / time step / PE
- *MADNESS*, time / reconstruction (precision)
- *Denovo*, time / transport step / unknown (cells,angles,moments,groups)

- *ScalaBLAST*, time / query / PE
- *DCA/QMC*, time / Green function update / slice
- LS3DF, time / DOF / SCF iteration
- *LAMMPS*, time / pairwise force / atom
- *Omega 3P*, time / eigenmode / PE
- *CCSM*, simulated years / wall clock day
- *VH-1*, time / zone update / PE
- *NCSM MC*, time / nucleon / shell / sample / img step
- *ENZO*, time / time step (depth) / PE

## CCSM CAM study metric details

- Parameters
  - Horizontal resolution : T85 spectral truncation (128x256)
  - Number of vertical levels: 26
  - Number of advected constituents: 3
  - Stable timestep: 10 min
  - Output interval: 1 month
  - CCSM CVS tag: cam3.0.19 vs cam3.2.19

- Application Based Observables / Metrics
  - Throughput: Simulated years per wall clock day : YPD
  - Average time (sec) in dynamics per day: dynpkg
  - Average time (sec) in physics/chemistry per day: physics
  - Average time (sec) in land model per day: land
  - Average time (sec) in dynamics- physics data transpose per day: dp
  - Average time (sec) in atm-land communication per day: cl2ck
  - Average time (sec) to simulate a day: stepon

# Performance is Limited by ...

1) **System power** -primary constraint (PUI, facility / total)

2) **Memory** bandwidth and capacity are not keeping pace

3) **Concurrency** 1000X increase in-node

4) **Processor** open question

5) **Programming model** compilers will not hide this

6) **Algorithms** need to minimize data movement, not flops

7) **I/O bandwidth** not on pace with machine speed

8) **Reliability and resiliency**

9) **Bisection bandwidth** limited by cost and energy

ref. ASCR exascale mtg

# *Need* metrics to quantify the data related costs on and across nodes

I

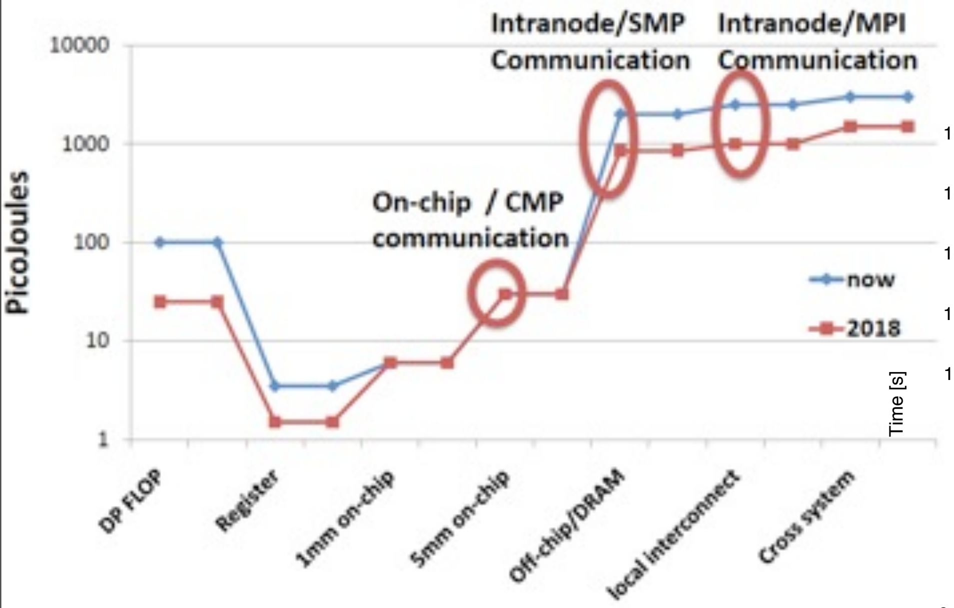
-refine performance measures for data movement and access costs as these dominate over floating point costs

- ***bandwidth***, the number of cycles a core waits because the bus is not ready; as the measure gets large, it indicates that the bus is in high demand and loads or stores involving main memory will take longer

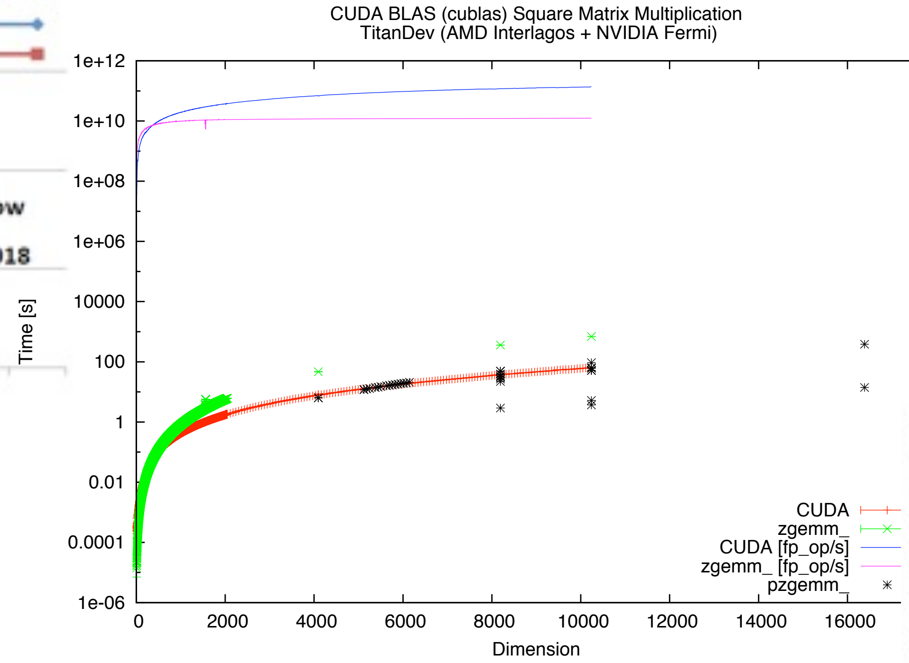
  -provides means to reason about performance costs versus (bisection) bandwidth scaling (i.e. increased node counts)

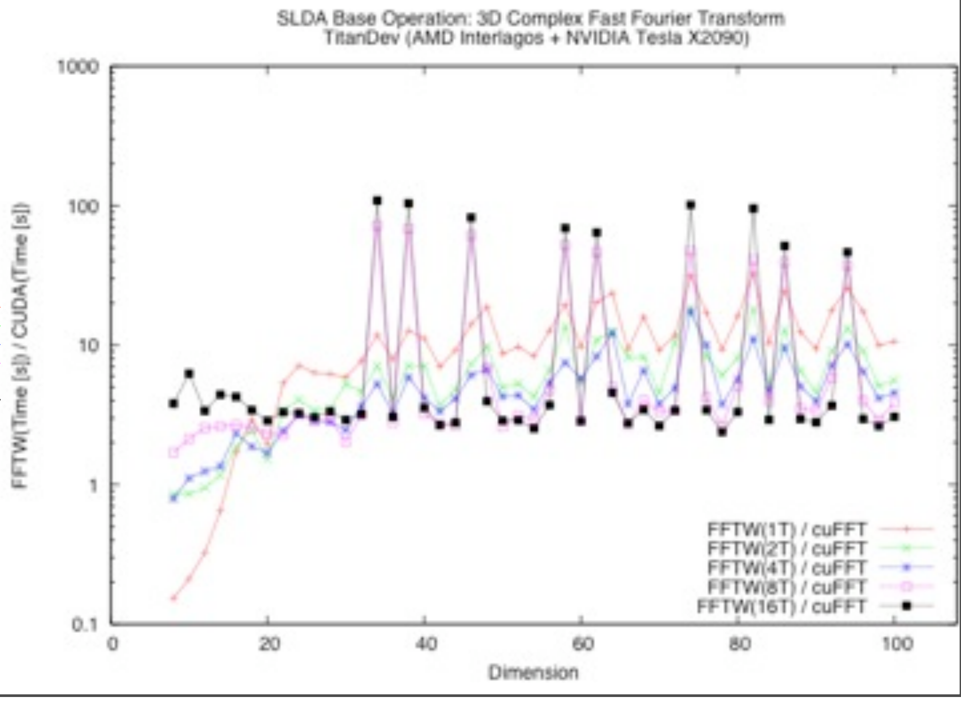
- ***locality***, the ratio of the peak versus measured capacity of each memory level (on/off chip) divided by access time in cycles

•i.e. consider ratio of gather and scatter costs in loops (A. Snavely, exascale planning meeting)



ref. ASCR exascale mtg





## *Need* extensions that relate performance to power; lead to novel optimization ideas

-extension of existing metrics to reason about power and performance tradeoffs, energy driven optimizations (i.e. DVFS)

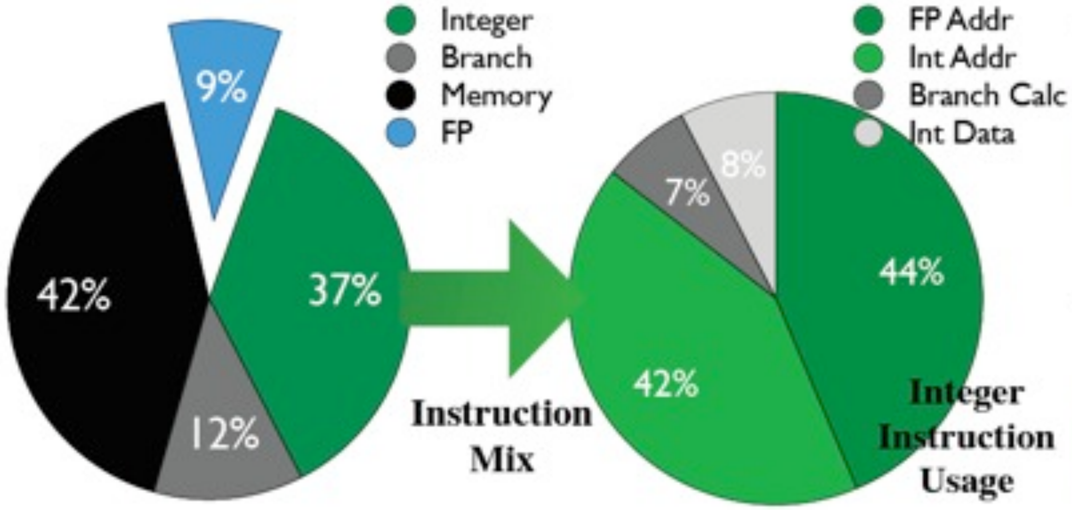
- -number of floating point operations per Watt (floating point dominated)
- -cost of loads or stores in bytes per Watt (data ops dominated)

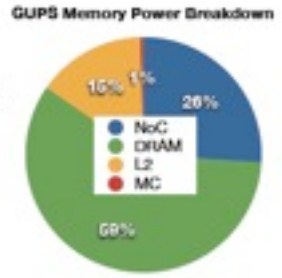
-metric guided optimizations to simultaneously minimize power consumption and time to solution (IBM Zurich study)

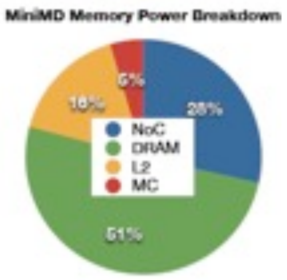
- -computational cost ~ *f(time to solution) * energy*
  - -*f* constant,  cost per execution event in Joules
  - -*f* linear, cost provides insight about appropriateness of hardware platform for application

-demand tools for power measurements

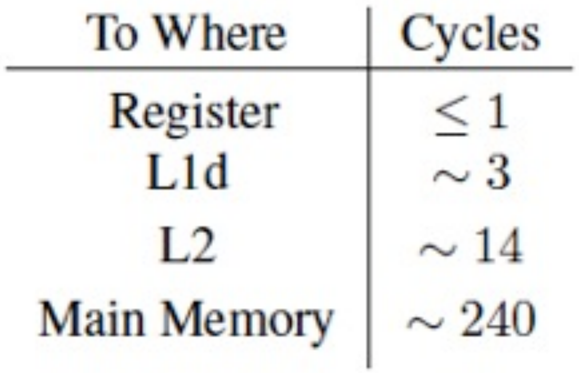
- -memory (29%), network (29%), floating point unit (16%))  (distribution of power in HPC hardware (Kogge))







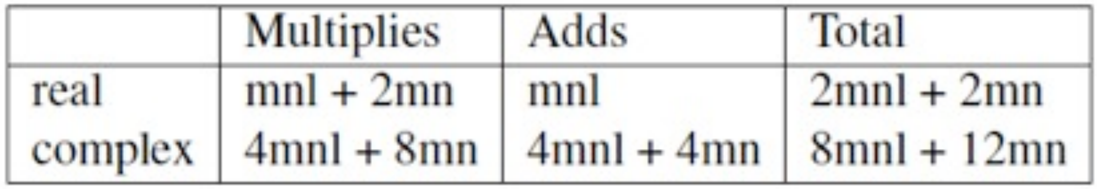
*-relate cycle costs in memory refs to energy in Joules*



| To Where | Cycles |
|---|---|
| Register | $\leq 1$ |
| L1d | $\sim 3$ |
| L2 | $\sim 14$ |
| Main Memory | $\sim 240$ |

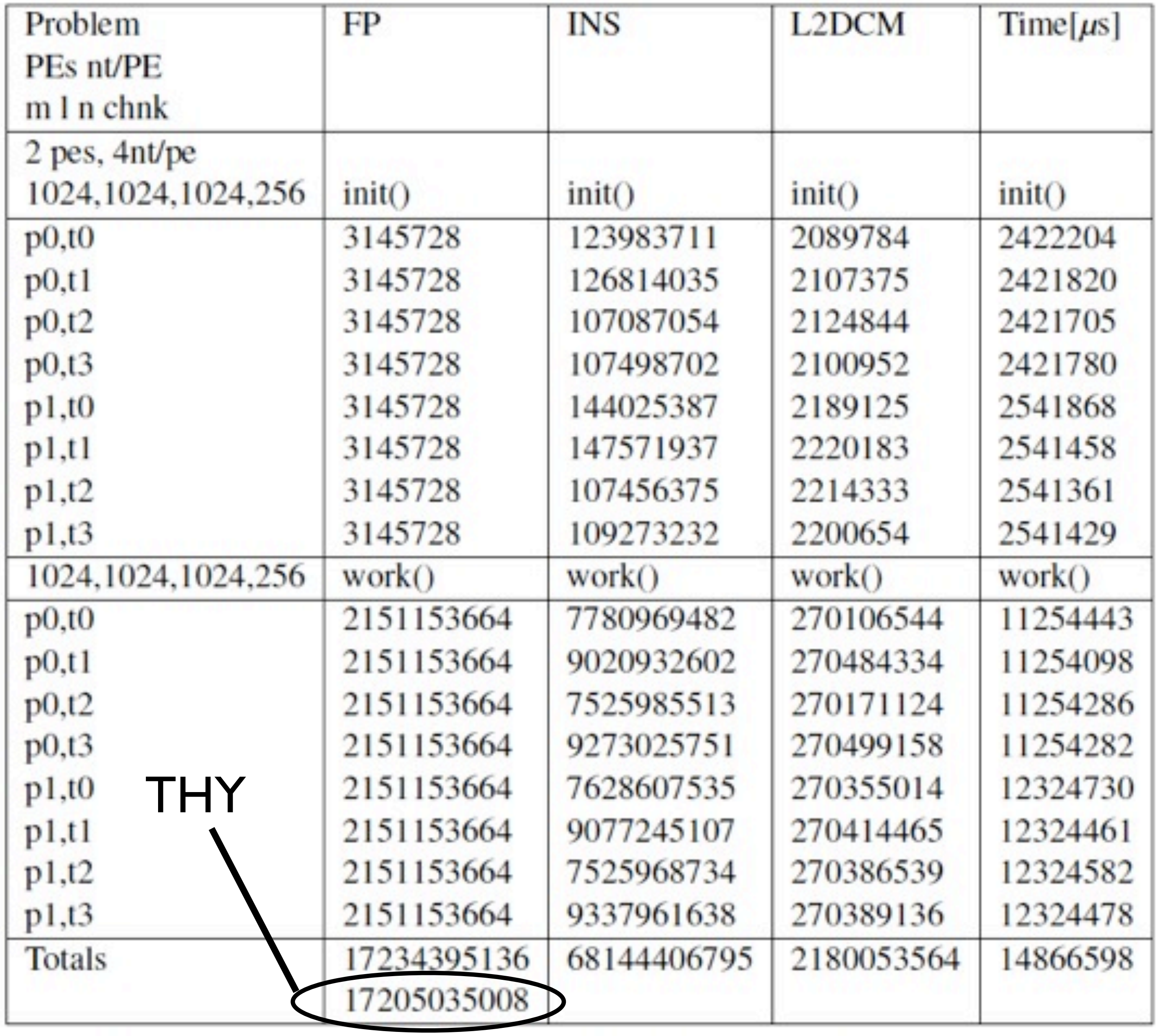
ref. ASCR exascale mtg

# *Need* (?) Accurate, Scalable Tools at *Thread Level*



| | Multiplies | Adds | Total |
|---|---|---|---|
| real | mnl + 2mn | mnl | 2mnl + 2mn |
| complex | 4mnl + 8mn | 4mnl + 4mn | 8mnl + 12mn |

Table 14: Theoretical complexity of $C(m,n) \leftarrow \alpha A(m,l)B(l,n) + \beta C(m,n)$.



| Problem<br>PEs nt/PE<br>m l n chnk | FP | INS | L2DCM | Time[$\mu$s] |
|---|---|---|---|---|
| 2 pes, 4nt/pe<br>1024,1024,1024,256 | init() | init() | init() | init() |
| p0,t0 | 3145728 | 123983711 | 2089784 | 2422204 |
| p0,t1 | 3145728 | 126814035 | 2107375 | 2421820 |
| p0,t2 | 3145728 | 107087054 | 2124844 | 2421705 |
| p0,t3 | 3145728 | 107498702 | 2100952 | 2421780 |
| p1,t0 | 3145728 | 144025387 | 2189125 | 2541868 |
| p1,t1 | 3145728 | 147571937 | 2220183 | 2541458 |
| p1,t2 | 3145728 | 107456375 | 2214333 | 2541361 |
| p1,t3 | 3145728 | 109273232 | 2200654 | 2541429 |
| 1024,1024,1024,256 | work() | work() | work() | work() |
| p0,t0 | 2151153664 | 7780969482 | 270106544 | 11254443 |
| p0,t1 | 2151153664 | 9020932602 | 270484334 | 11254098 |
| p0,t2 | 2151153664 | 7525985513 | 270171124 | 11254286 |
| p0,t3 | 2151153664 | 9273025751 | 270499158 | 11254282 |
| p1,t0 | 2151153664 | 7628607535 | 270355014 | 12324730 |
| p1,t1 | 2151153664 | 9077245107 | 270414465 | 12324461 |
| p1,t2 | 2151153664 | 7525968734 | 270386539 | 12324582 |
| p1,t3 | 2151153664 | 9337961638 | 270389136 | 12324478 |
| Totals | 17234395136<br>17205035008 | 68144406795 | 2180053564 | 14866598 |

Table 17: Measured machine events of threaded parallel work phase (zgemm).

1 PE, 4 nt / PE
Group / Function / Thread (max)
==============================
Total
---------------------------------------------
Time% 100.0%
Time 12.213947 secs

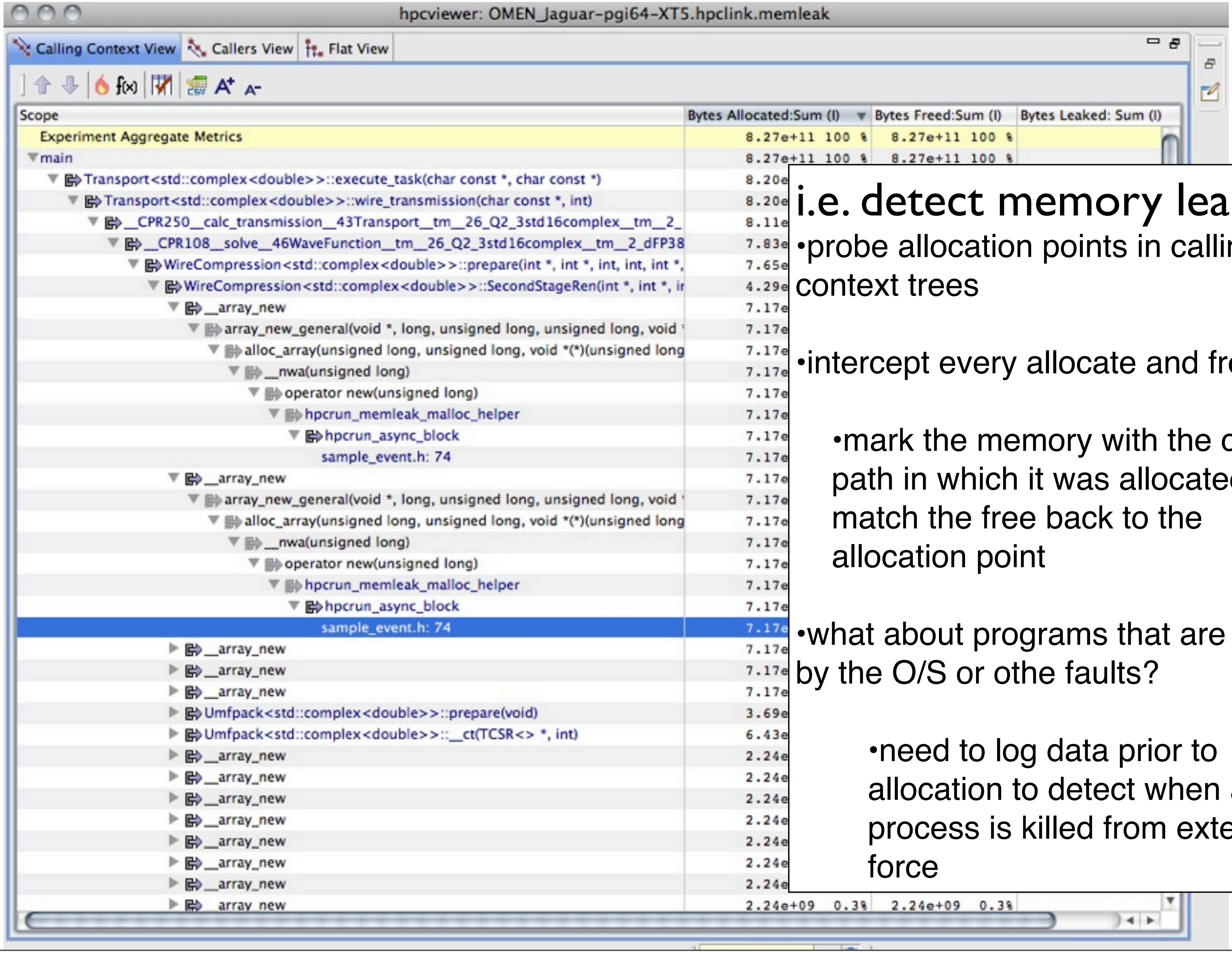
TOT_INS
1037.779M/sec
10063040357 instr

FP_INS
222.330M/sec
2155872263 ops (2154299392)

TOT_CYC
9.697 secs
21332826724 cycles

User time (approx) 100.0% Time
12.214 secs
26870760748 cycles

# *Need* (?) Accurate, Scalable Memory Tools



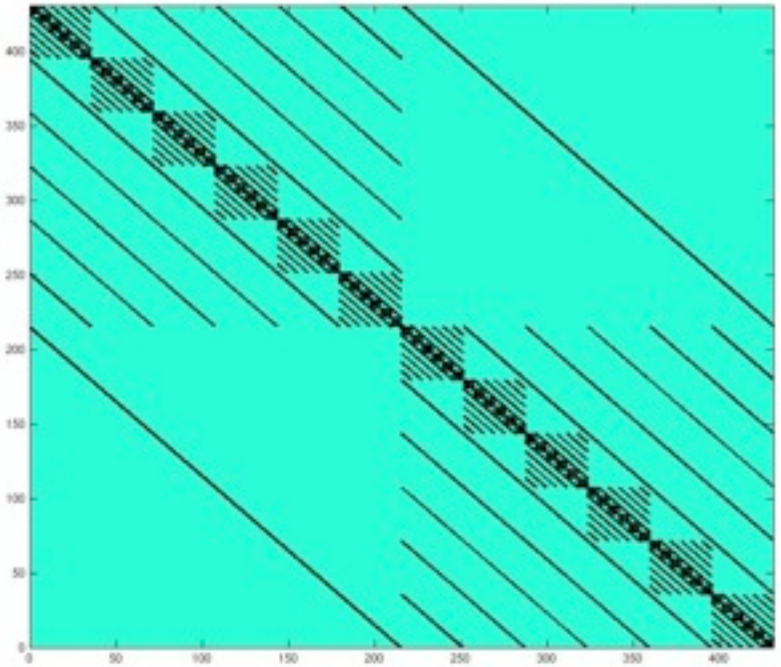
| Scope | Bytes Allocated:Sum (I) | Bytes Freed:Sum (I) | Bytes Leaked: Sum (I) |
|---|---|---|---|
| Experiment Aggregate Metrics | 8.27e+11 100 % | 8.27e+11 100 % | |
| main | 8.27e+11 100 % | 8.27e+11 100 % | |
| Transport<std::complex<double>>::execute_task(char const *, char const *) | 8.20e | | |
| Transport<std::complex<double>>::wire_transmission(char const *, int) | 8.20e | | |
| __CPR250__calc_transmission__43Transport__tm__26_Q2_3std16complex__tm__2_ | 8.11e | | |
| __CPR108__solve__46WaveFunction__tm__26_Q2_3std16complex__tm__2_dFP38 | 7.83e | | |
| WireCompression<std::complex<double>>::prepare(int *, int *, int, int, int *, | 7.65e | | |
| WireCompression<std::complex<double>>::SecondStageRen(int *, int *, ir | 4.29e | | |
| __array_new | 7.17e | | |
| array_new_general(void *, long, unsigned long, unsigned long, void | 7.17e | | |
| alloc_array(unsigned long, unsigned long, void *(*)(unsigned long | 7.17e | | |
| __nwa(unsigned long) | 7.17e | | |
| operator new(unsigned long) | 7.17e | | |
| hpcrun_memleak_malloc_helper | 7.17e | | |
| hpcrun_async_block | 7.17e | | |
| sample_event.h: 74 | 7.17e | | |
| __array_new | 7.17e | | |
| array_new_general(void *, long, unsigned long, unsigned long, void | 7.17e | | |
| alloc_array(unsigned long, unsigned long, void *(*)(unsigned long | 7.17e | | |
| __nwa(unsigned long) | 7.17e | | |
| operator new(unsigned long) | 7.17e | | |
| hpcrun_memleak_malloc_helper | 7.17e | | |
| hpcrun_async_block | 7.17e | | |
| sample_event.h: 74 | 7.17e | | |
| __array_new | 7.17e | | |
| __array_new | 7.17e | | |
| __array_new | 7.17e | | |
| Umfpack<std::complex<double>>::prepare(void) | 3.69e | | |
| Umfpack<std::complex<double>>::__ct(TCSR<> *, int) | 6.43e | | |
| __array_new | 2.24e | | |
| __array_new | 2.24e | | |
| __array_new | 2.24e | | |
| __array_new | 2.24e | | |
| __array_new | 2.24e | | |
| __array_new | 2.24e | | |
| __array_new | 2.24e | | |
| __array_new | 2.24e+09 0.3% | 2.24e+09 0.3% | |

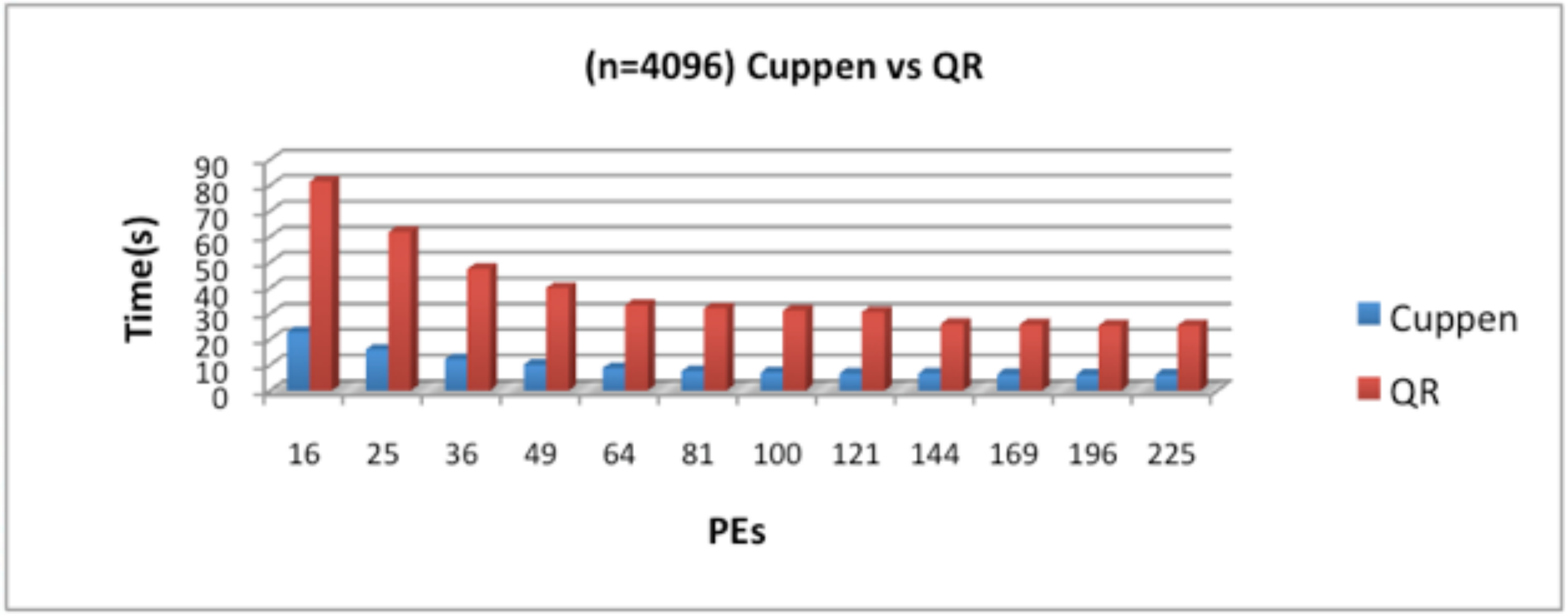
## i.e. detect memory leaks

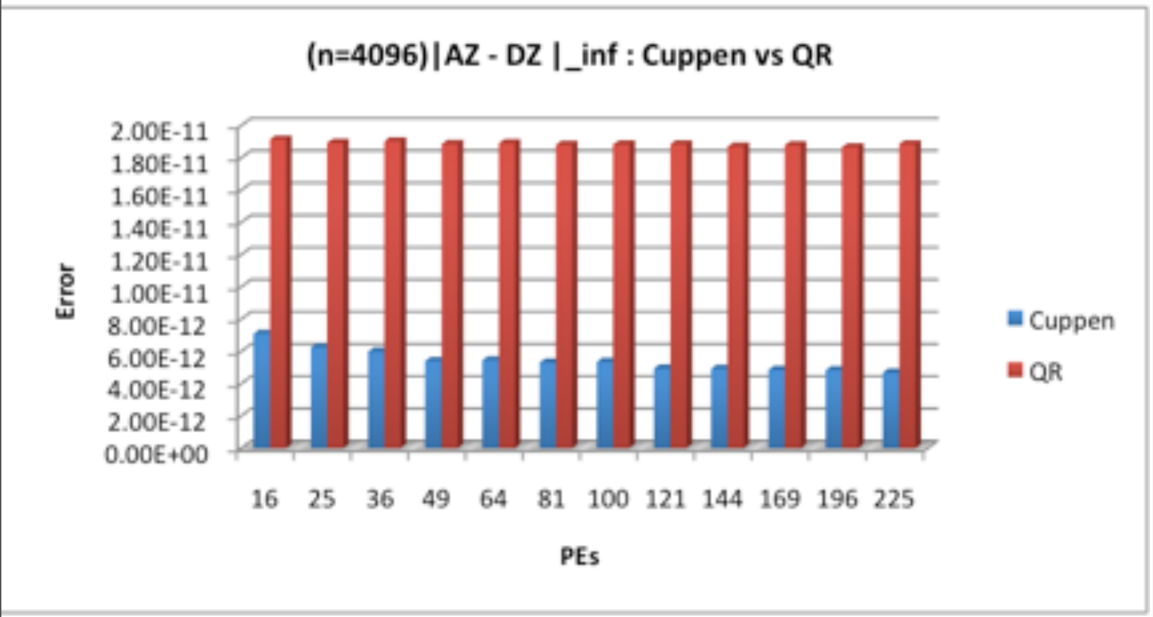
- probe allocation points in calling context trees
- intercept every allocate and free
  - mark the memory with the call path in which it was allocated, match the free back to the allocation point
- what about programs that are killed by the O/S or othe faults?
  - need to log data prior to allocation to detect when a process is killed from external force

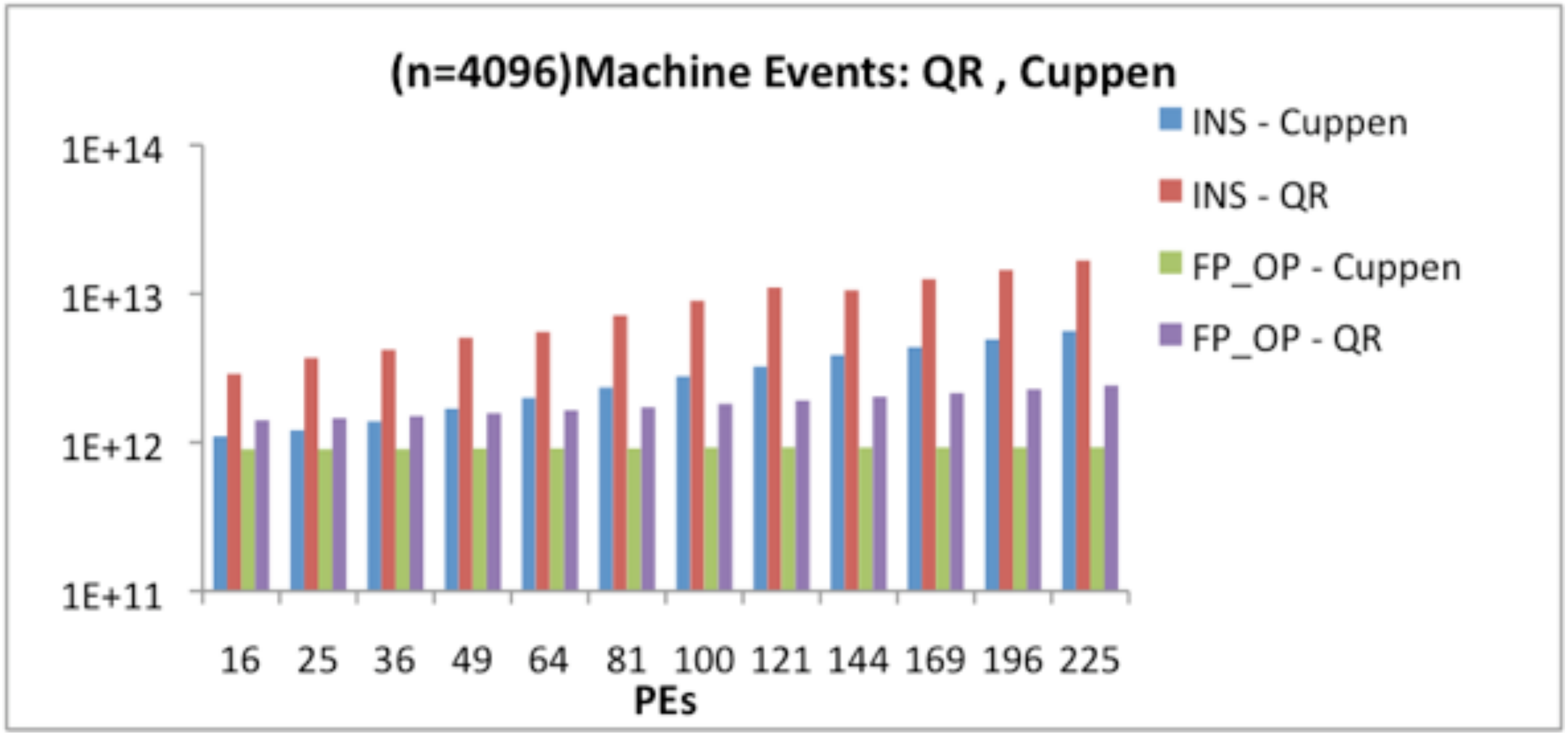
# *Need* algorithms that Improve {ins,*flop(s)*} / *byte* (and don't compromise accuracy or performance)

•J.J.M. Cuppen, *A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem*, Numer. Math. 36, 177-195 (1981)
•F. Tisseur and J.J. Dongarra, *Parallelizing the Divide and Conquer Algorithm for the Symmetric Tridiagonal Eigenvalue Problem on Distributed Memory Architectures*, lawn132 (1998)









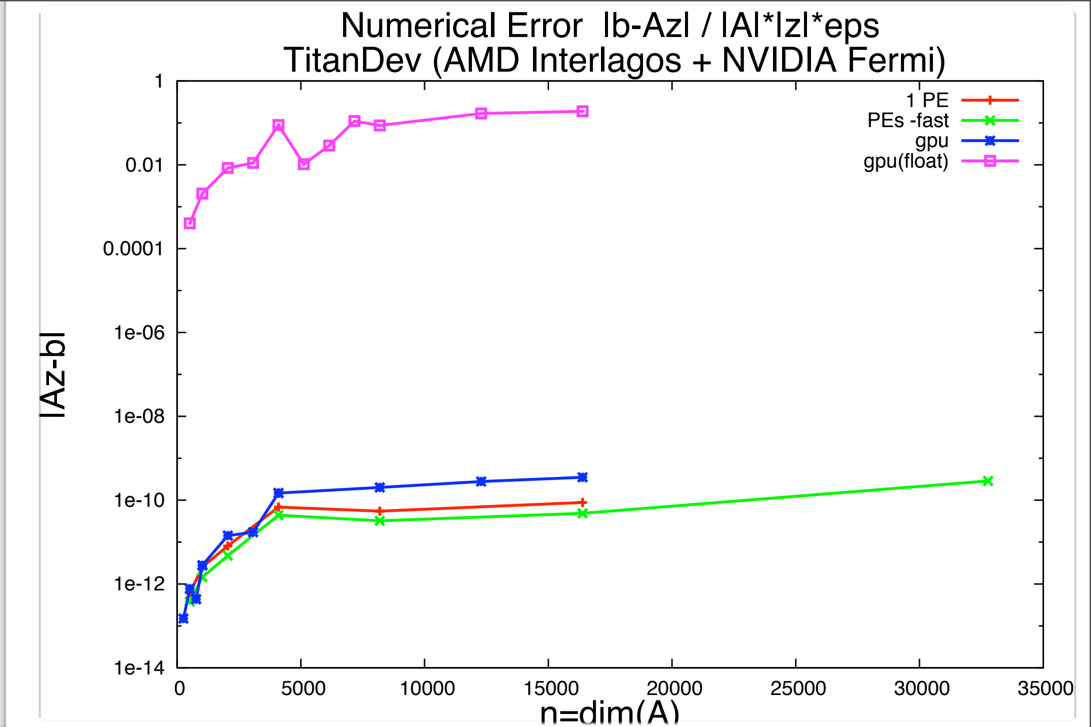
# Mixed Precision Solvers -faster, controlled accuracy

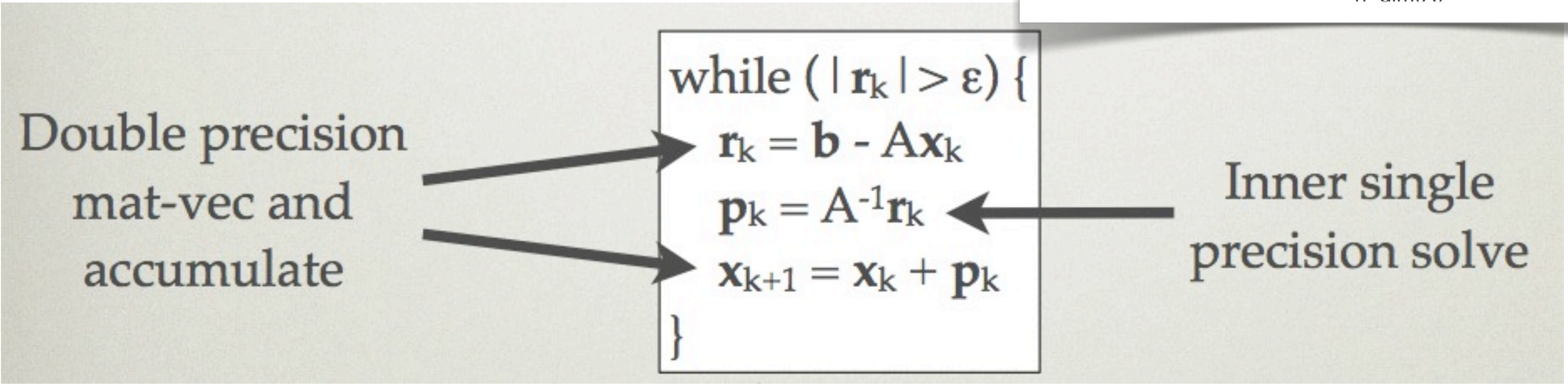
Require solver tolerance beyond limit of single precision

- DP is at least 2X slower
- Use iterative refinement

Double precision done can be done on CPU or GPU

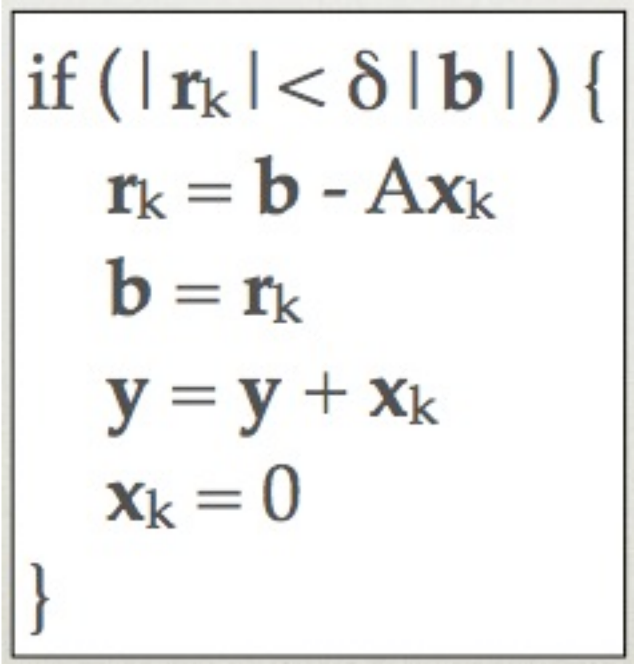
Disadvantage is each new single precision solve is a restart


Numerical Error |b-Az| / |A|*|z|*eps
TitanDev (AMD Interlagos + NVIDIA Fermi)


Double precision mat-vec and accumulate

```
while ($|r_k| > \varepsilon$) {
    $r_k = b - Ax_k$
    $p_k = A^{-1}r_k$
    $x_{k+1} = x_k + p_k$
}
```

Inner single precision solve



```
if ($|r_k| < \delta |b|$) {
    $r_k = b - Ax_k$
    $b = r_k$
    $y = y + x_k$
    $x_k = 0$
}
```

Reliable Updates (Sleijpen and Van der Worst 1996)

- Iterated residual diverges from true residual
- Occasionally replace iterated residual with true residual
- Also use second accumulator for solution vector

Single-precision can be used to find double-precision result

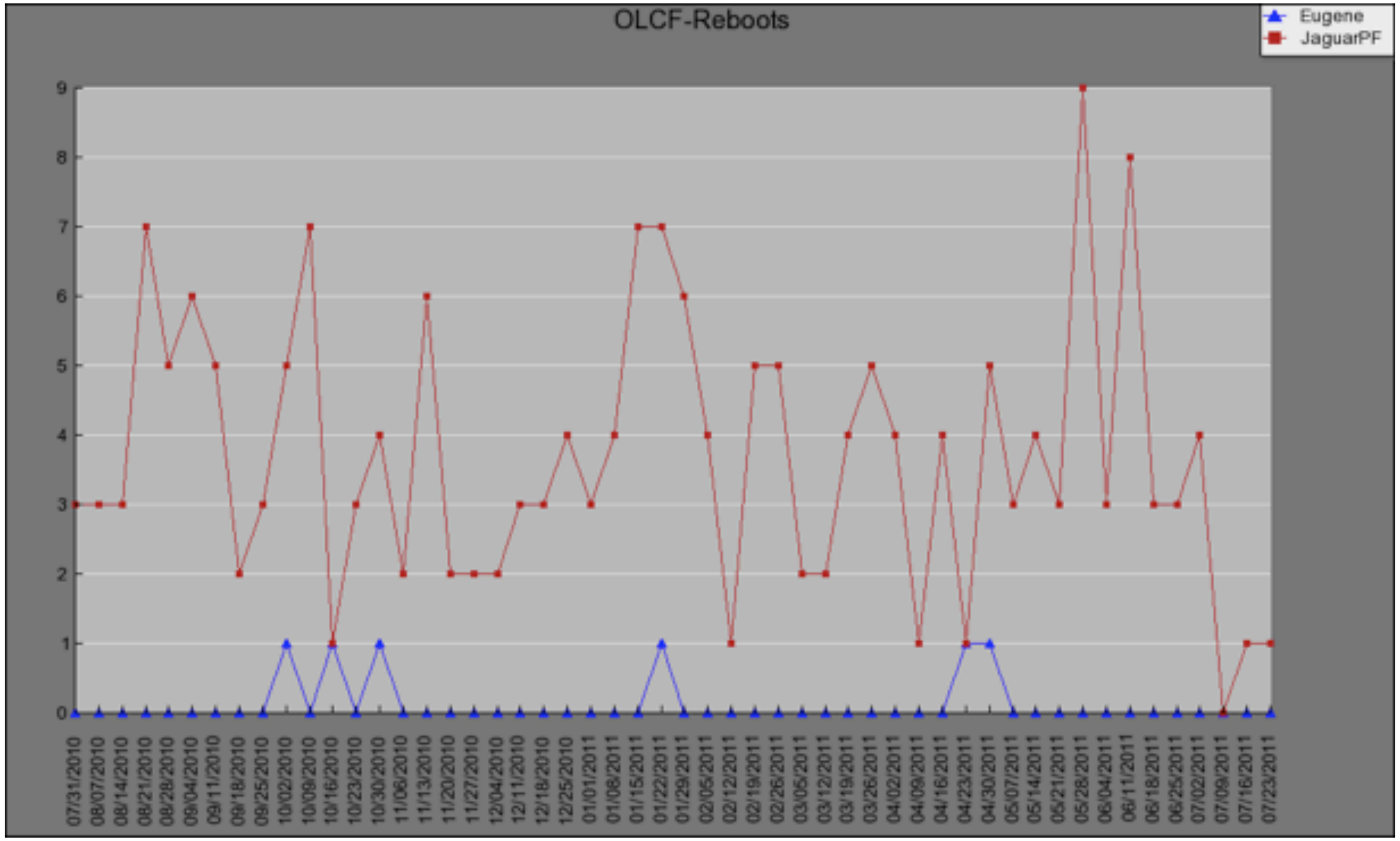
- GPU kernel is still bandwidth bound
- Use half precision for inner solve?

ref. M. Clark, NVIDIA
my PD, Saul Cohen -multigrid

# *Need* measures for detecting, mitigating, recovering from failures

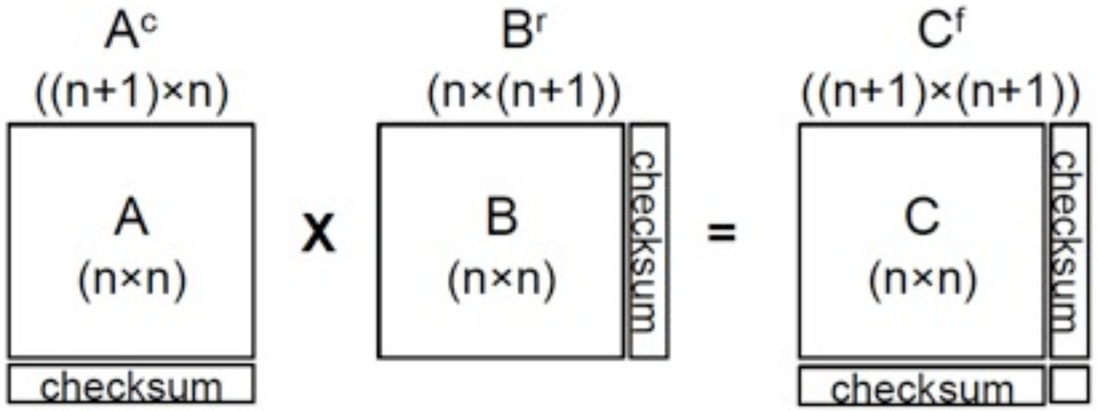
- fail / continue
- hard / soft faults

- resiliency must go beyond check point / restart
  - algorithm based fault tolerance



COSTS
/ TRADEOFFS ?

have to go beyond single failure



$$\mathbf{A}^c_{n+1,j} = \sum_{i=1}^{n} \mathbf{A}_{ij}$$

$$\mathbf{B}^r_{i,n+1} = \sum_{j=1}^{n} \mathbf{B}_{ij}$$

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \text{ and } \mathbf{C}^f = \mathbf{A}^c * \mathbf{B}^r$$

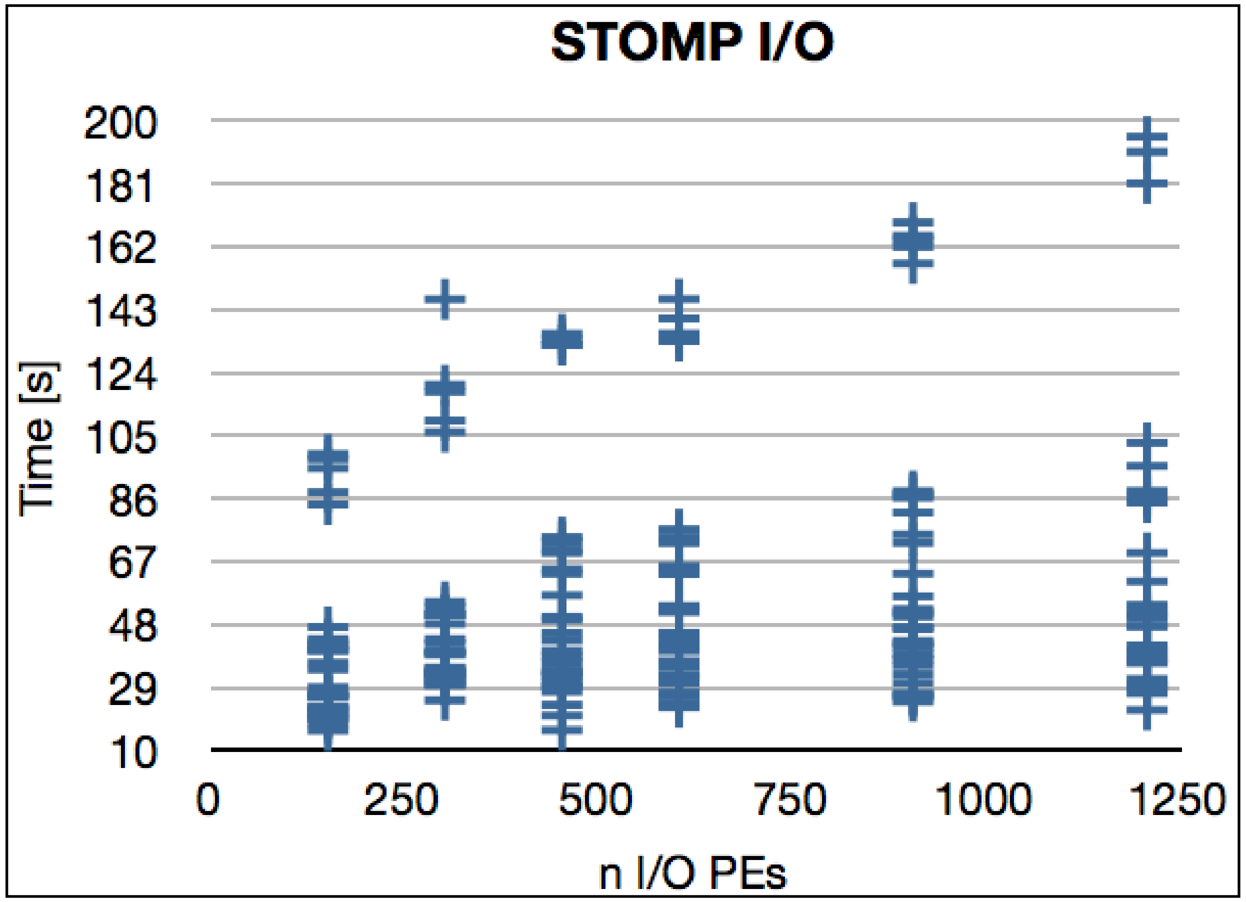
$$\mathbf{C}^f_{n+1,j} = \sum_{i=1}^{n} \mathbf{C}^f_{ij} \quad \mathbf{C}^f_{i,n+1} = \sum_{j=1}^{n} \mathbf{C}^f_{ij}$$

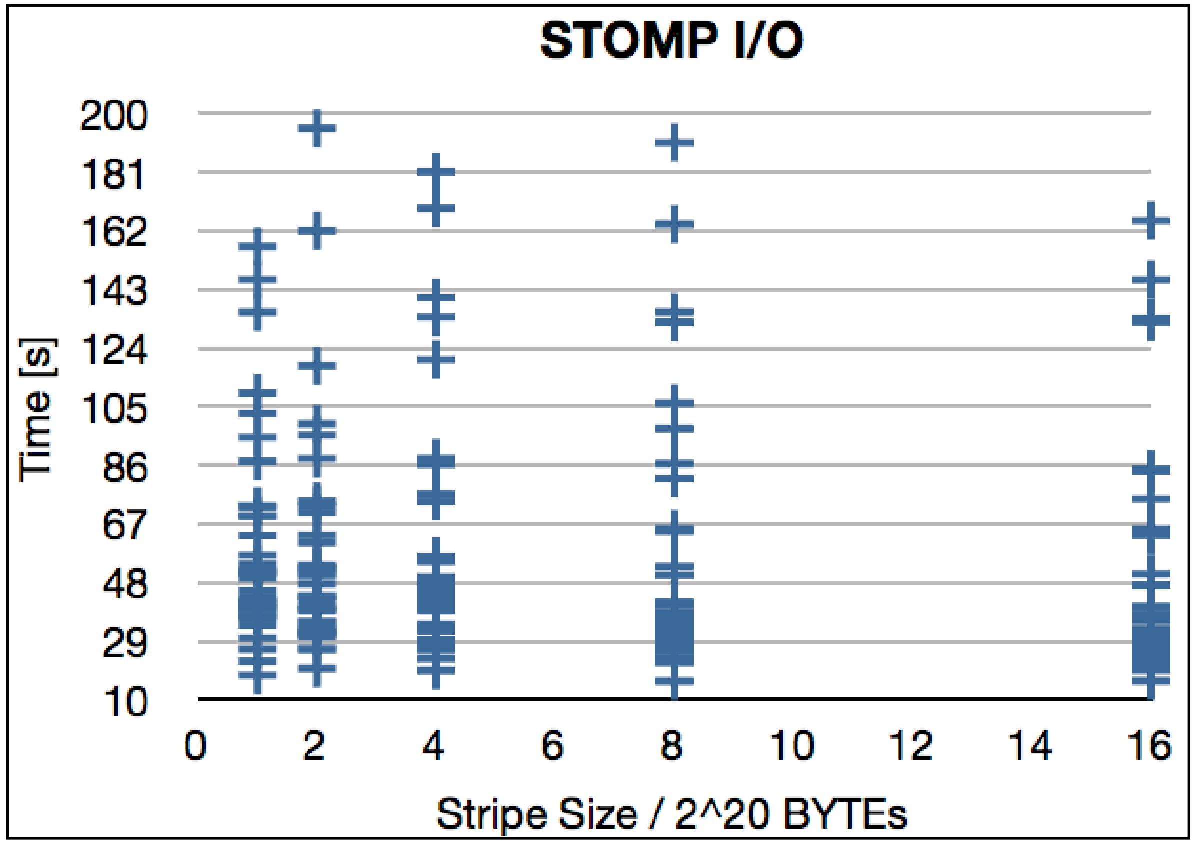
ref P. Raghavan's work

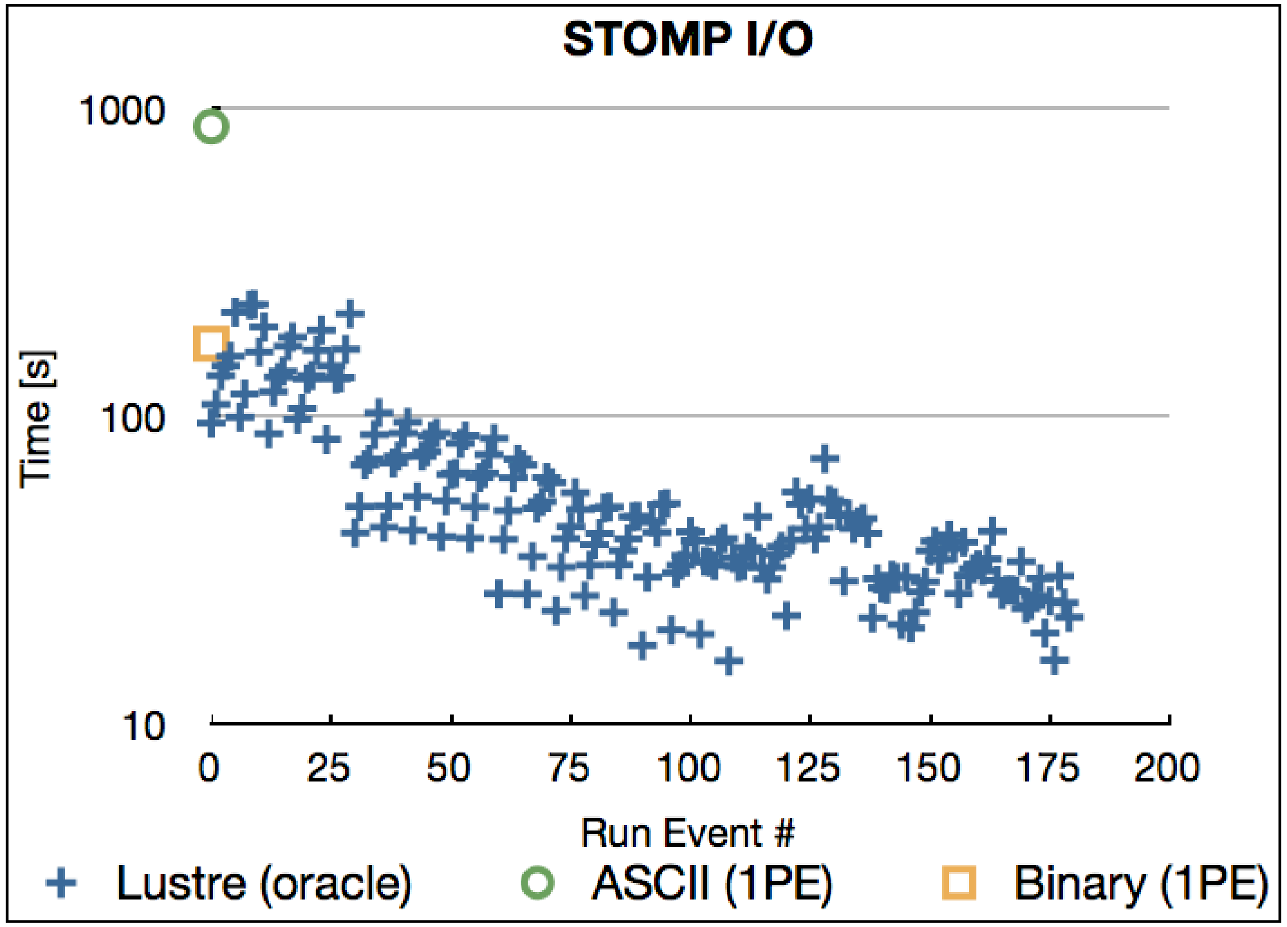
# *Need* measures for I/O operations for applications

Parameters set in the file system related to but independent from the problem parameters:

- Number of OSTs
  1, 2, 4, 8, 16, 32
- Stripe size in BYTEs
  1 MB, 2 MB, 4MB, 8 MB, 16 MB
- access pattern (round robin)
- Number of I/O PEs for spatial decomposition
  kio ~ 1, 2, 3, 4, 6, 8
- Total number of I/O PEs is kio * nfld
  since nfld =151, 151, 302, 453, 604, 906, 1208







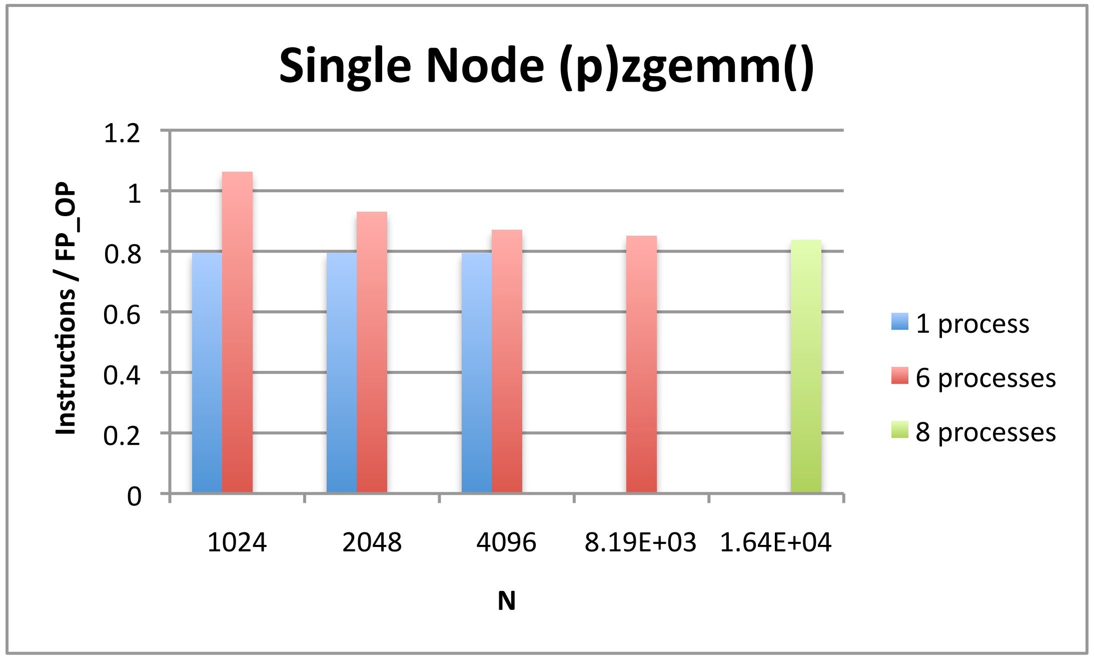
# Apps Are Not Usually Dominated by FLOPs

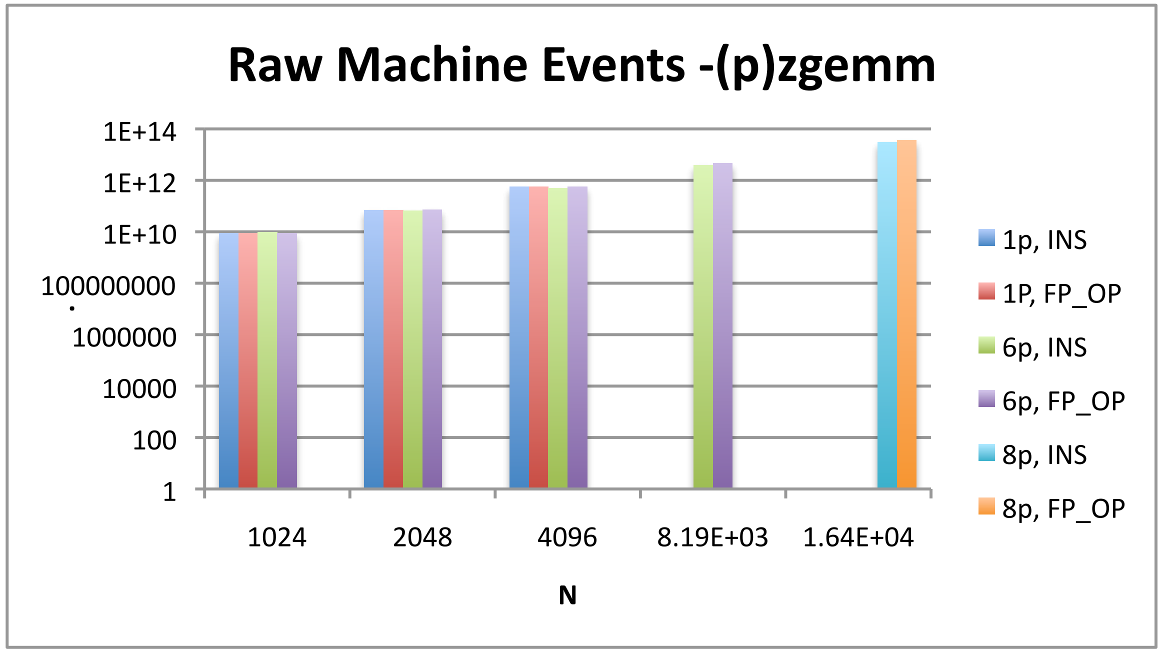
| Application | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Instructions Retired | 1.99E+15 | 8.69E+17 | 1.86E+19 | 2.45E+18 | 1.24E+16 | 7.26E+16 | 8.29E+18 | 2.67E+18 |
| Floating Point Ops | 3.52E+11 | 1.27E+15 | 1.95E+18 | 2.28E+18 | 6.16E+15 | 4.15E+15 | 3.27E+17 | 1.44E+18 |
| INS / FP_OP | 5.64E+03 | 6.84E+02 | 9.56 | 1.08 | 2.02 | 17.5 | 25.3 | 1.85 |

**REFERENCE FLOATING POINT INTENSE PROBLEM :: Dense Matrix Matrix Multiplication**

C <--- a A B + b C :: OPERATIONAL COMPLEXITY :A[m,n] , B[n,p] , C[m,p] :: [ 8mpn + 13mp ] FLOP

E.g. m=n=p=1024 ---> 8603566080 FLOP , measure 8639217664

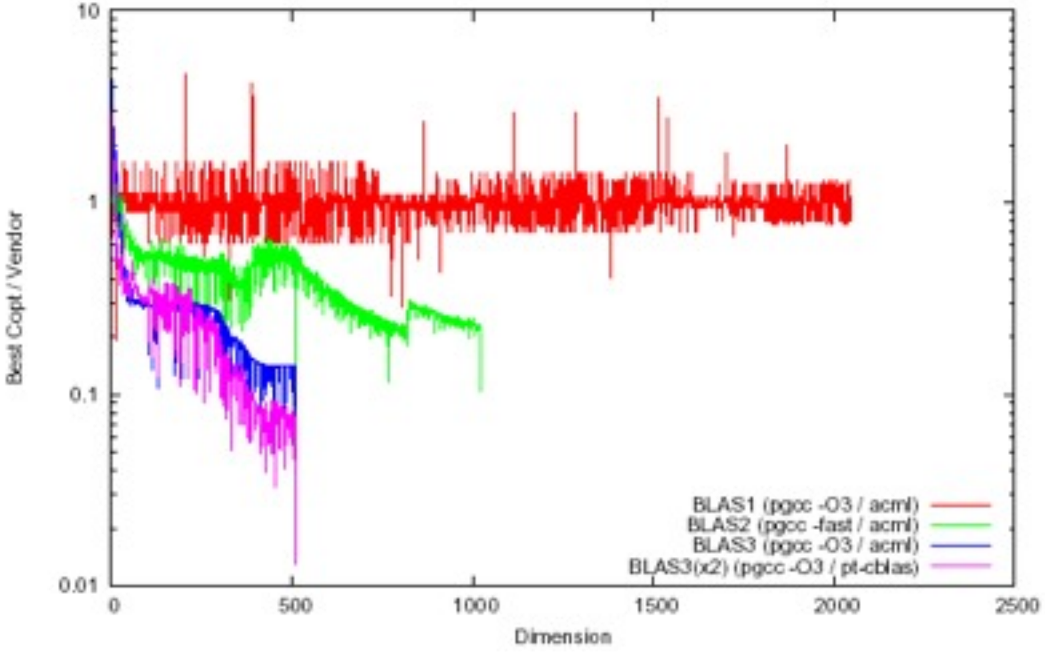




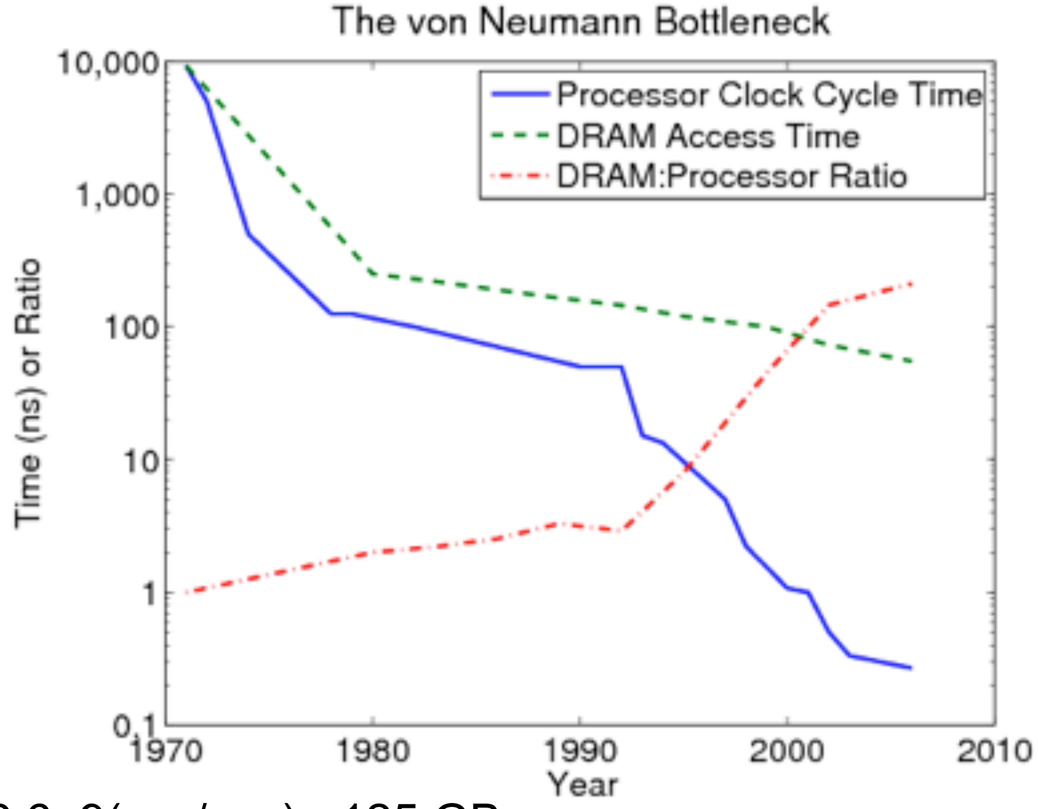
# Memory Wall *Always* There ...

**Computation**: Theoretical peak: (# cpu cores) * (flops / cycle / core) * (cycles / second)

**Memory**: Theoretical peak: (bus width) * (bus speed)



**BLAS 1:** O(n) operations on O(n) operands
**BLAS 2:** O(n**2) operations on O(n**2) operands
**BLAS 3:** O(n**3) operations on O(n**2) operands



*y* = α *x* + *y* :

3 loads, 1 store
(more expensive than FP_OPs by a long shot)
2 floating point operations (maybe 1) on 3 operands

eg, double precision on the FY10 target platform:

(3 operands / 2 flop ) * (8 bytes / operand) * 6 core * 4 ( flop / cyc /core) * 2.6e9(cyc/sec) ~125 GBps

... We don’t have this and to get it is $$$ ... how to achieve **Sustainability??**

# Memory Issues

CPU waiting for memory hierarchy is bottom line of idle time

- memory latency
- miss rates
  - instruction stalls
  - branch misprediction
  - unresolved data dependencies

O/S stall times are substantial cost -not easily influenced

- misses
- coping with interference from the application
  - write references: how big should the write buffer be + queuing model

# Memory Measurements

- first touches are expensive
  - misses lead to repeated first touches
  - repeated dynamic allocation / free lead to first touches
- costs can be measured

accesses / second (access rate)

x

N_{fractional miss ratio} / access

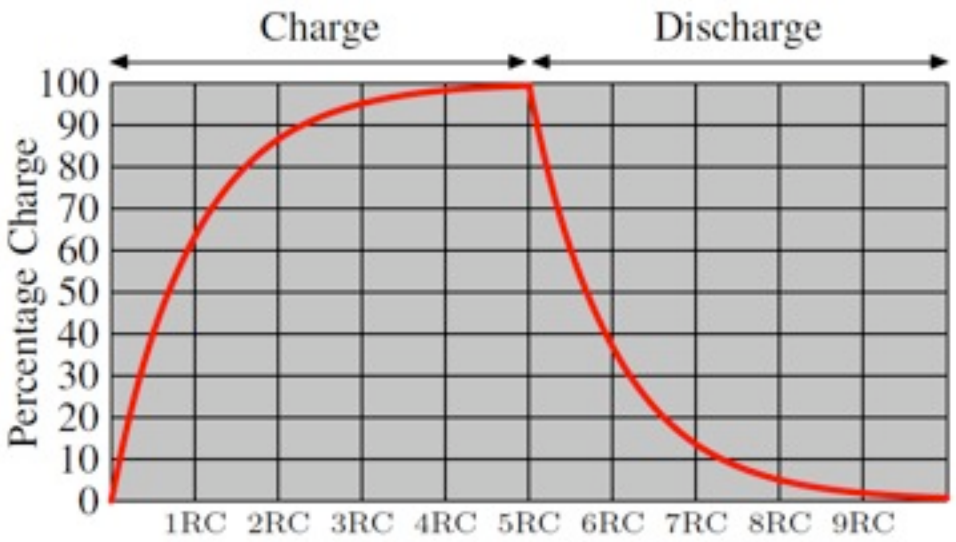
x

bytes / miss

:= bytes / second

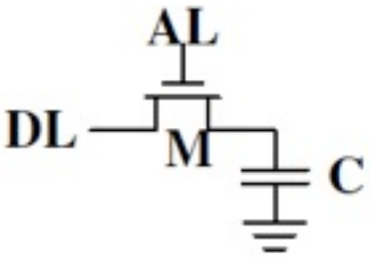
- but, to be accurate requires average memory access times over the duration of program execution
- a program’s locality behavior is not constant during execution and is basically unique

# DRAM COSTS: Power = Capacity * Voltage^2 * Frequency



**DRAM**
**C**, capacitor, keeps cell state

**M**, transistor, controls access to cell state



**read** the state of the cell the **access line AL** is raised
-causes a current to flow on the data line DL or not

**write** to the cell the **data line DL** is appropriately set and AL is raised for a time long enough to charge or drain the capacitor

## Today's Memories ...

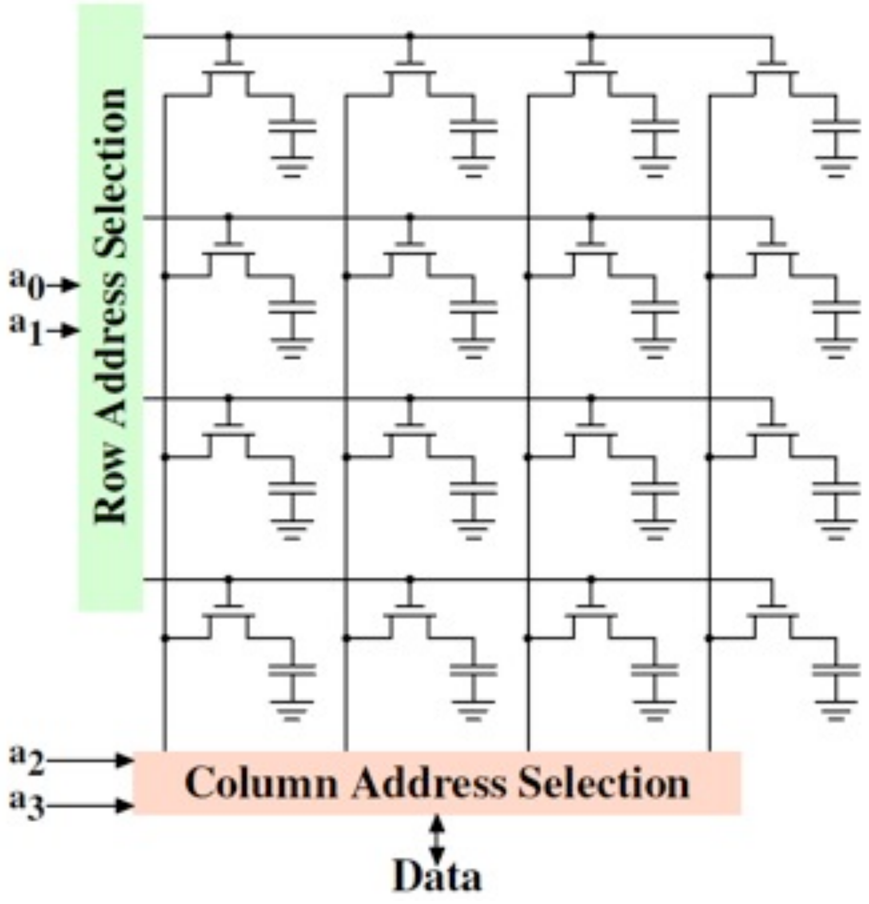
- 10^9 cells
- cell capacitance < femto-farad
- resistance O(tera-ohms)

## Refresh Cycles ~ 64ms

- leakage
- reading drains the charge (read + recharge)
- stall cycle on bus > 11 cpu cycles

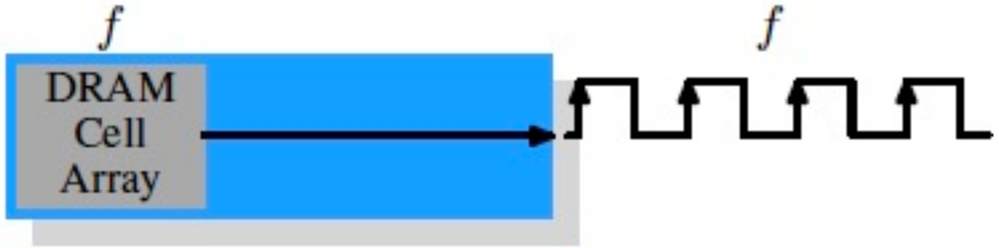
## Faster memory

- lower voltage --> decreases stability,
- increase frequency --> $$$ as arrays get large
- (i.e. more addressable memory) and voltage is increased to assure stability

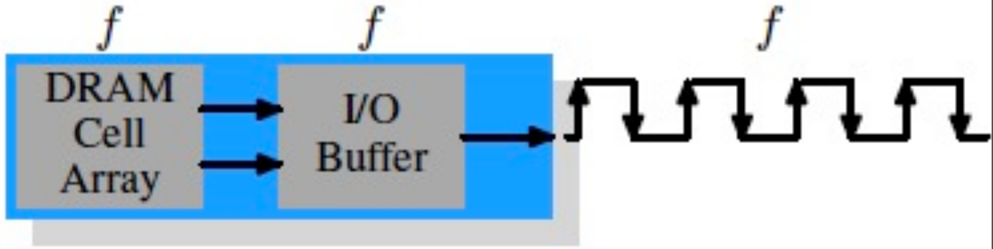


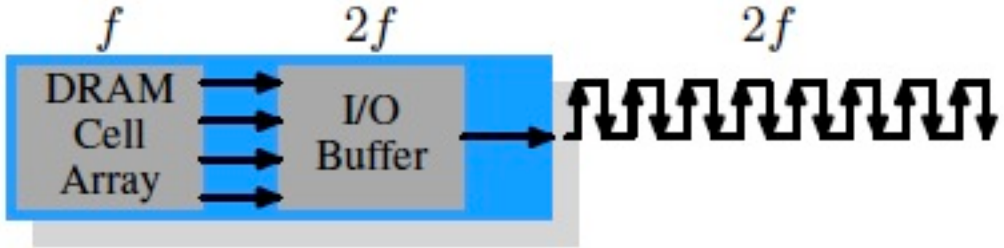
ref. Drepper, What every Programmer Should Know about Memory

**SDR (PC100)** ~
DRAM cell array 100MHz
data transfer rate 100Mbps



**DDR (PC1600)** ~ moves 2X the data / clock (leading , falling)
add “I/O” buffer (2 bits on data line) adjacent to DRAM cell array
pull two adjacent column cells per access over 2 line data bus
100 MHz X 64 bit / data bus X 2 data bus lines = 1600 MBps

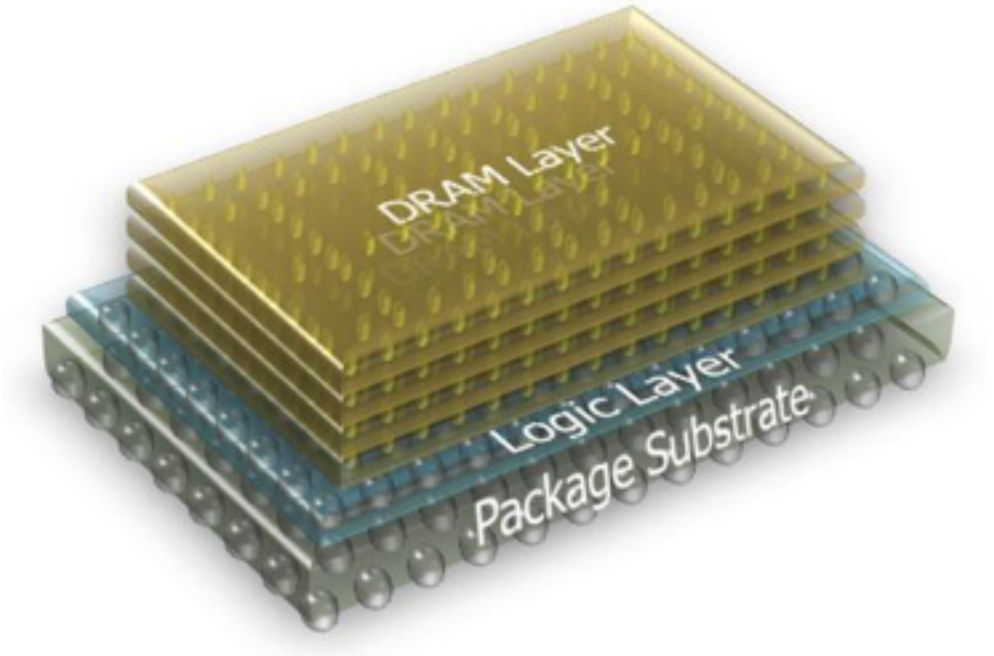


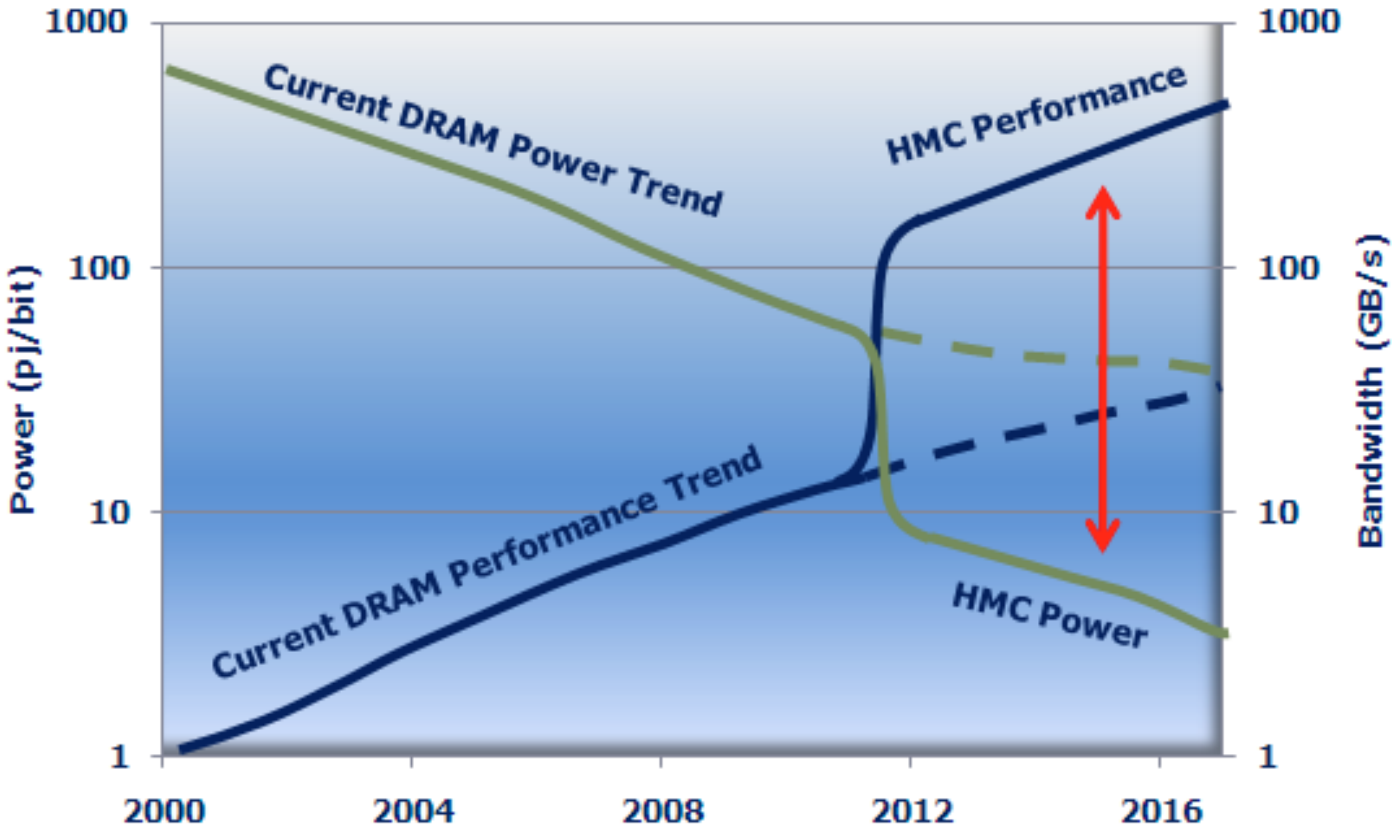


**DDR2 (PC6400)** ~ moves 4X the data / clock
double the bus frequency --> 2X bandwidth
double “I/O” buffer speed to match the bus
4 bits / clock on 4 line data bus
200MHz array; 400MHz bus; 800MHz FSB (effective freq)
200 MHz X 64 bit / data bus X 4 data bus lines = 6400 MBps
240 PIN addressing @ 1.8V

***each stall cycle on the memory bus is > 11 cpu cycles even in the best systems**

# HOW to DEAL w/ new Technologies: Micron Hybrid Memory Cube





***Reduced latency*** – With vastly more responders built into HMC, we expect lower queue delays and higher bank availability, which can provide a substantial system latency reduction, which is especially attractive in network system architectures.

***Increased bandwidth*** — A single HMC can provide more than **15x the performance of a DDR3** module. Speed is increased by the very fast, innovative interface, unlike the slower parallel interface used in current DRAM modules.

***Power reductions*** — HMC is exponentially more efficient than current memory, using **70% less energy per bit than DDR3**.

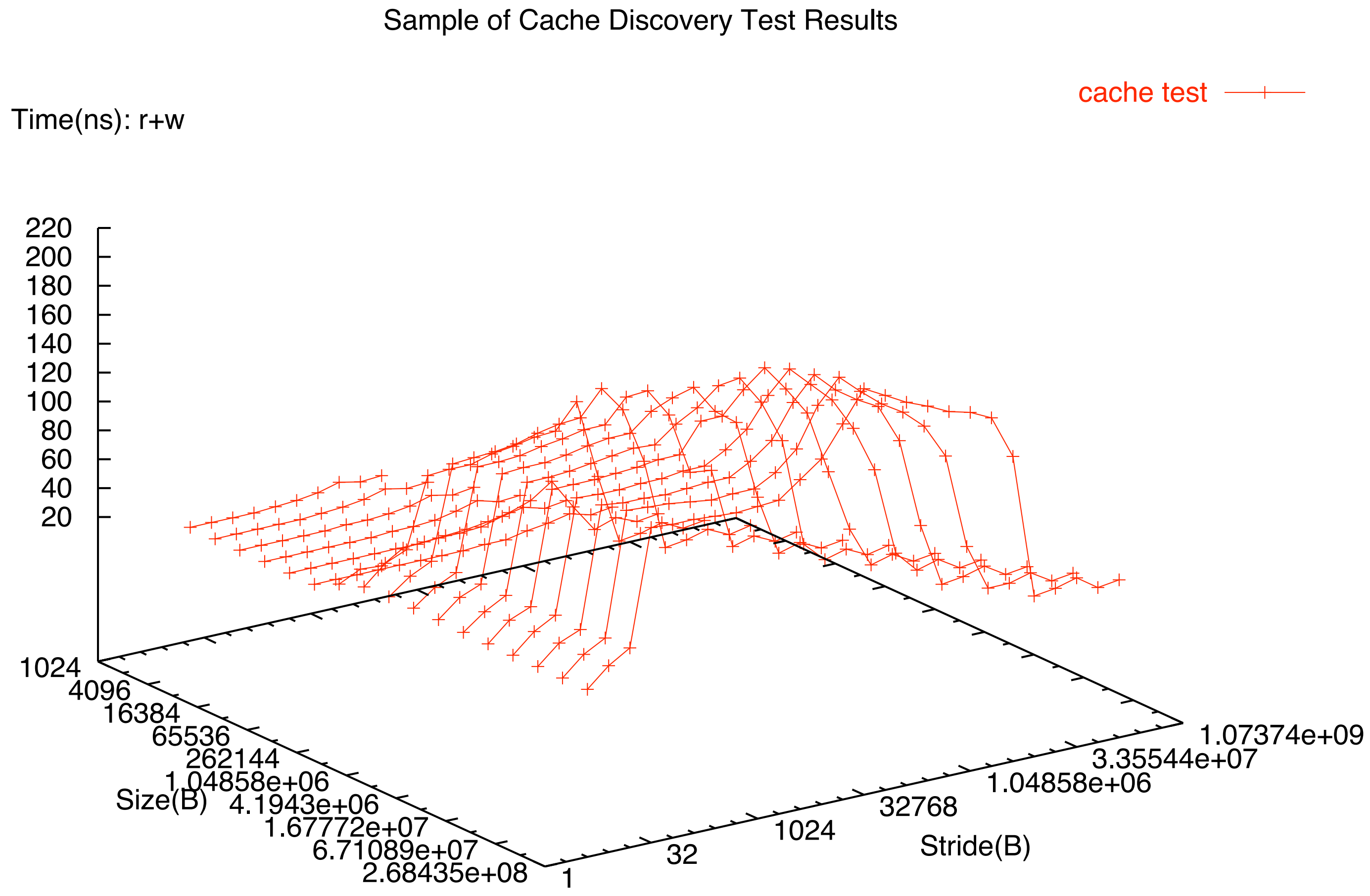
***Smaller physical systems*** — HMC's stacked architecture uses nearly **90% less space than today's RDIMMs**.

***Pliable to multiple platforms*** — Logic-layer flexibility allows HMC to be tailored to multiple platforms and applications.

# Hierarchical caches to hide memory latencies

**temporal locality**
when a referenced resource is referenced again sometime in the near future

**spatial locality**
the chance of referencing a resource is higher if a resource near it was just referenced



## Cache Coherency:

**write-through**, if cache line is written to, the processor also writes to main memory (at all times cache and memory are synchronized)

**write-back**, cache line is marked dirty, write back is delayed to when cache line is being evicted

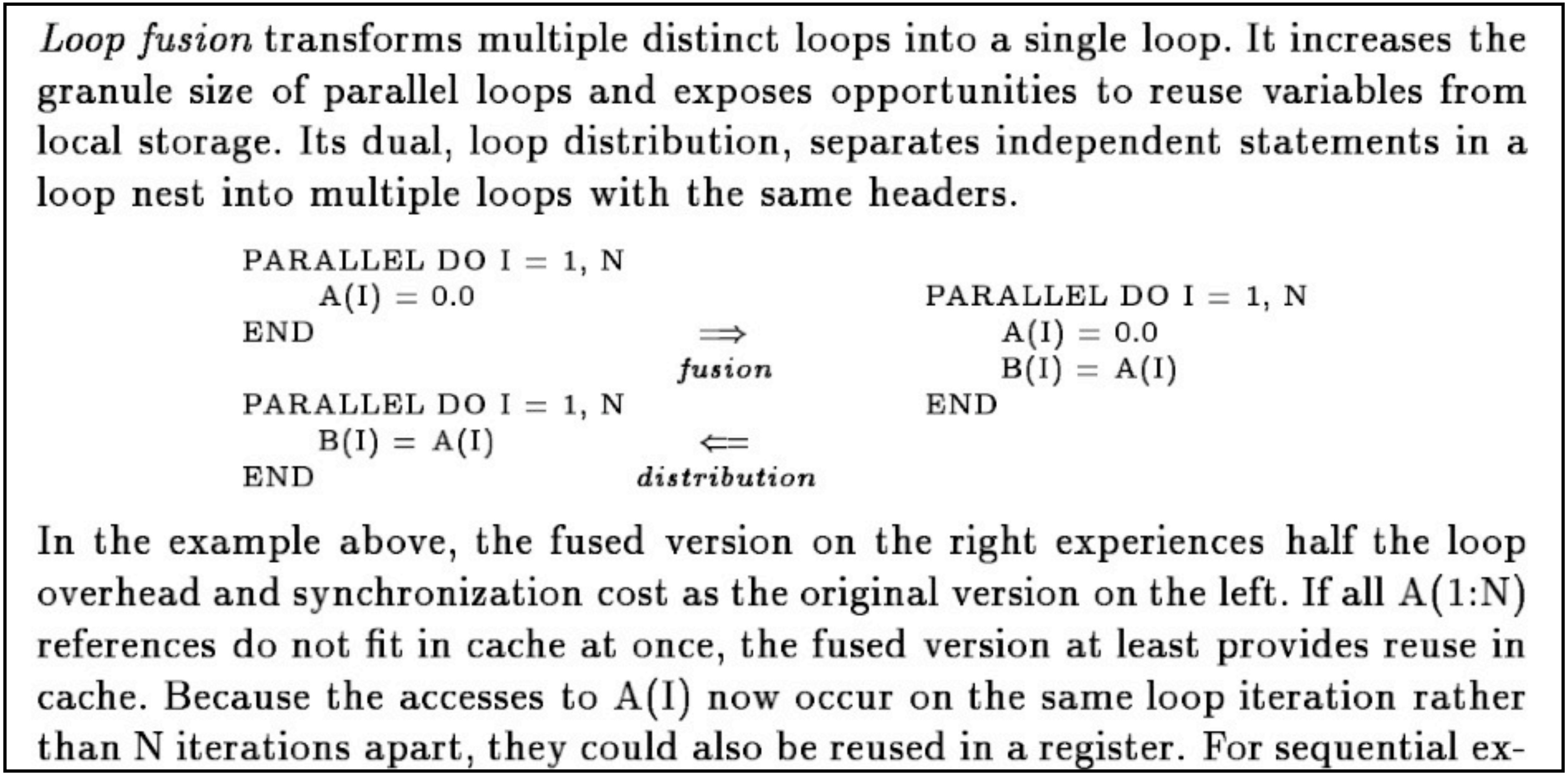
>1 processor core is active (say in SMP) -all processors still have to see the same memory content; have to exchange CL when needed -includes the MC

**write-combining** (ie on graphics cards)

# Use of Cache Inspired Basic Optimizations

base / node focus

- **non-temporal writes,** ie don't cache the data writes since it won't be used again soon (i.e. n-tuple initialization)
  - avoids reading cache line before write, avoids wasteful occupation of cache line and time for write (memset()); does not evict useful data
  - sfence() compiler set barriers
- **loop unrolling** , transposing matrices
- **vectorization**, 2,4,8 elements computed at the same time (SIMD) w/ multi-media extensions to ISA
- **reordering** elements so that elements that are used together are stored together -pack CL gaps w/ usable data (i.e. try to access structure elements in the order they are defined in the structure)
- **stack alignment,** as the compiler generates code it actively aligns the stack inserting gaps where needed ... is not necessarily optimal -if statically defined arrays, there are tools that can improve the alignment; separating n-tuples may increase code complexity but improve performance
- **function inlining**, may enable compiler or hand -tuned instruction pipeline optimization (ie dead code elimination or value range propagation) ; especially true if a function is called only once
- **prefetching**, hardware, tries to predict cache misses -with 4K page sizes this is a hard problem and costly penalty if not well predicted; software (void _mm_prefetch(void *p, enum _mm_hint h) --_MM_HINT_NTA -when data is evicted from L1d -don't write it to higher levels)



*Loop fusion* transforms multiple distinct loops into a single loop. It increases the granule size of parallel loops and exposes opportunities to reuse variables from local storage. Its dual, loop distribution, separates independent statements in a loop nest into multiple loops with the same headers.

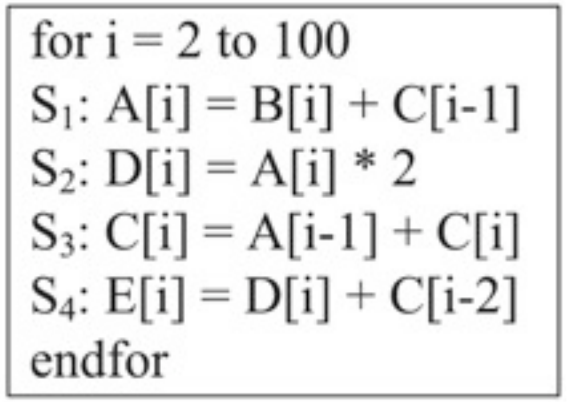
```
PARALLEL DO I = 1, N
      A(I) = 0.0                                PARALLEL DO I = 1, N
END                         ⟹                         A(I) = 0.0
                          fusion                      B(I) = A(I)
PARALLEL DO I = 1, N                            END
      B(I) = A(I)           ⟸
END                     distribution
```

In the example above, the fused version on the right experiences half the loop overhead and synchronization cost as the original version on the left. If all A(1:N) references do not fit in cache at once, the fused version at least provides reuse in cache. Because the accesses to A(I) now occur on the same loop iteration rather than N iterations apart, they could also be reused in a register. For sequential ex-

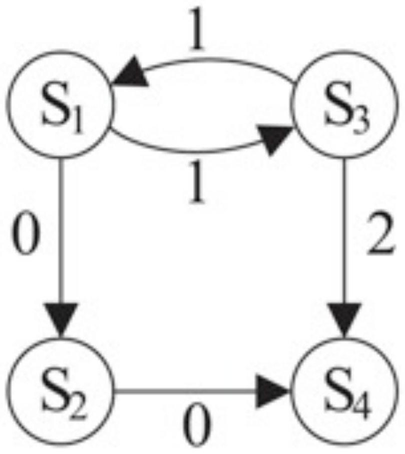
source: K. Kennedy, *Rice*

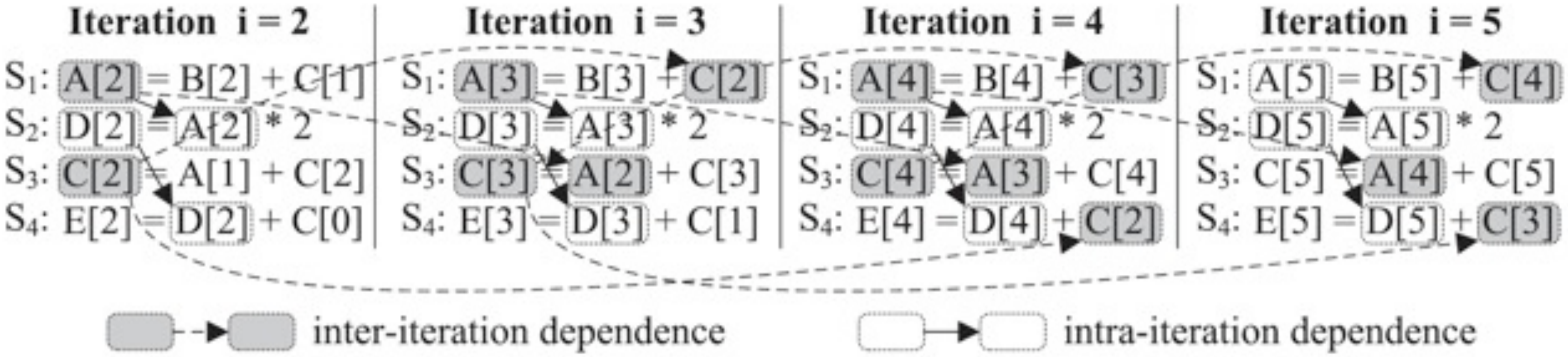
- reduce synchronization overheads in parallel loops
- improve data locality

# Going Beyond Instruction Level //ism to Loop Level



```
for i = 2 to 100
S1: A[i] = B[i] + C[i-1]
S2: D[i] = A[i] * 2
S3: C[i] = A[i-1] + C[i]
S4: E[i] = D[i] + C[i-2]
endfor
```





| Iteration i = 2 | Iteration i = 3 | Iteration i = 4 | Iteration i = 5 |
|---|---|---|---|
| S1: A[2] = B[2] + C[1] | S1: A[3] = B[3] + C[2] | S1: A[4] = B[4] + C[3] | S1: A[5] = B[5] + C[4] |
| S2: D[2] = A[2] * 2 | S2: D[3] = A[3] * 2 | S2: D[4] = A[4] * 2 | S2: D[5] = A[5] * 2 |
| S3: C[2] = A[1] + C[2] | S3: C[3] = A[2] + C[3] | S3: C[4] = A[3] + C[4] | S3: C[5] = A[4] + C[5] |
| S4: E[2] = D[2] + C[0] | S4: E[3] = D[3] + C[1] | S4: E[4] = D[4] + C[2] | S4: E[5] = D[5] + C[3] |

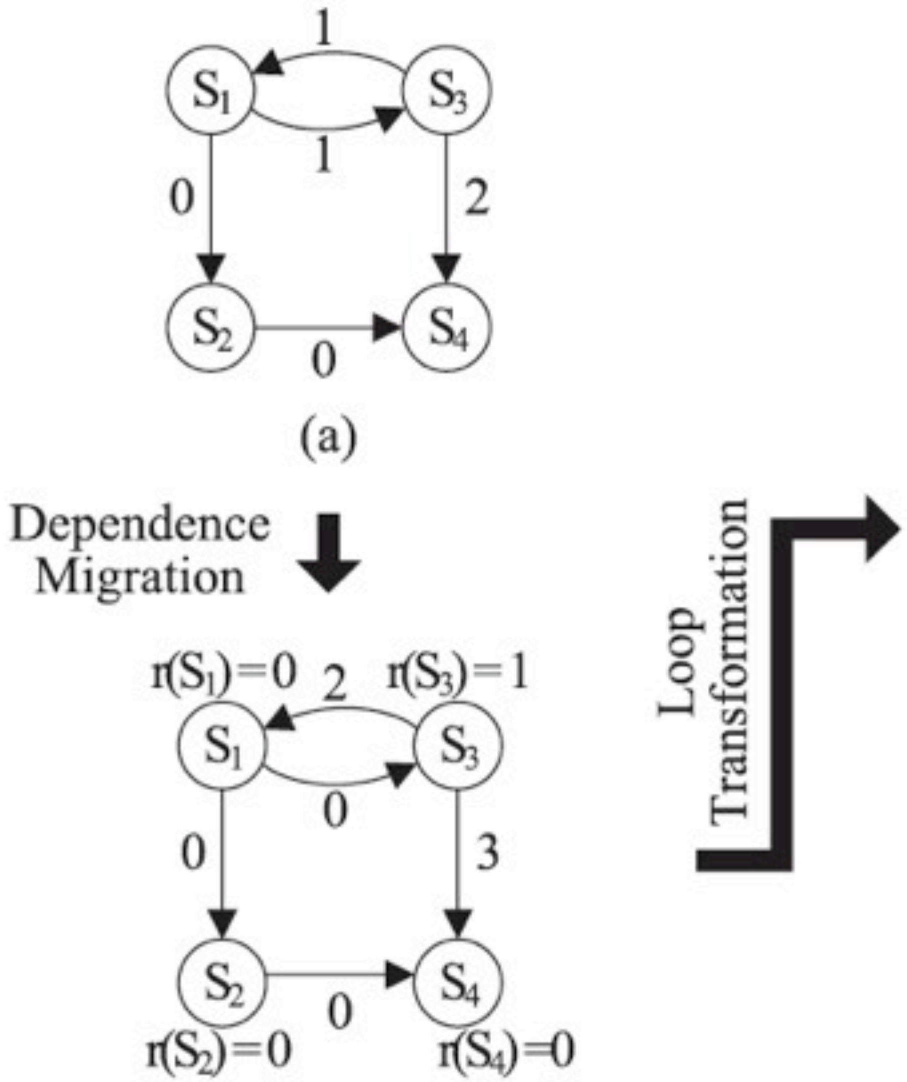
▨ → ▨ inter-iteration dependence

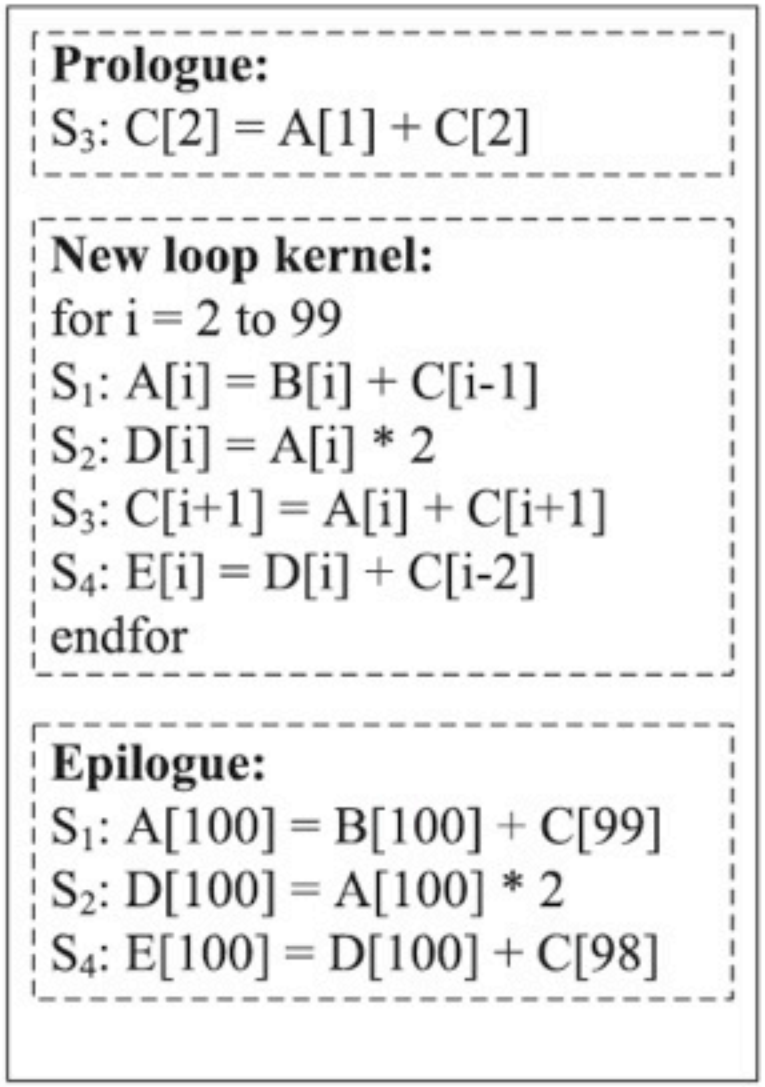
□ → □ intra-iteration dependence

***before***, minimum nonzero edge weight = 1

***after***, minimum nonzero edge weight = 2

(re)moving dependencies decreases stalls, decreases time





**Prologue:**

```
S3: C[2] = A[1] + C[2]
```

**New loop kernel:**

```
for i = 2 to 99
S1: A[i] = B[i] + C[i-1]
S2: D[i] = A[i] * 2
S3: C[i+1] = A[i] + C[i+1]
S4: E[i] = D[i] + C[i-2]
endfor
```

**Epilogue:**

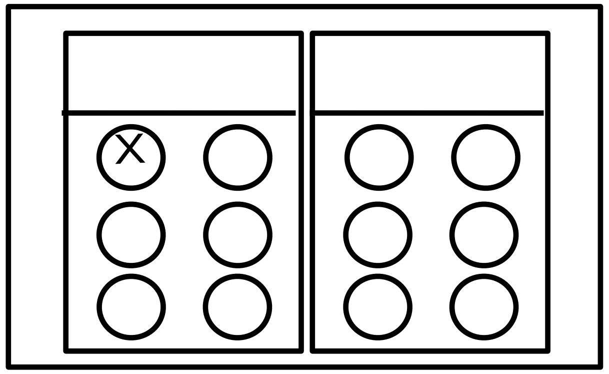
```
S1: A[100] = B[100] + C[99]
S2: D[100] = A[100] * 2
S4: E[100] = D[100] + C[98]
```

Optimally Maximizing Iteration-Level Loop Parallelism, D. Liu et al., IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 23, NO. 3, MARCH 2012

# Challenge: Exploit Multi-core Hybrid / New Programming Methods

**task based**: Cilk for instance
**other:** Chapel

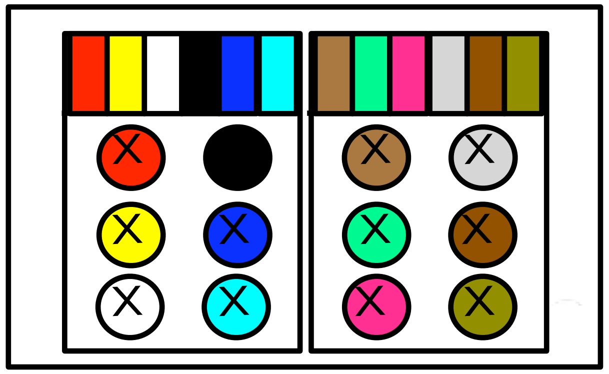
- **MPI, processes** spawn lightweight processes
- **PGAS,** partitioned global address space
- **OpenMP threads**, directive based, #include <omp.h> , omp_set_num_threads();
- **POSIX threads**, #include <pthread.h> , pthread_create();
- **NVIDIA's CUDA C,** kernel execution (distinct executables)
- **OpenCL,** (abstract processing elements, compute units and devices -heterogeneous systems)
- **OpenACC**, directive / pragma based, compiler creates kernel for execution on GPU (based on PGI Accelerator, M. Wolfe)



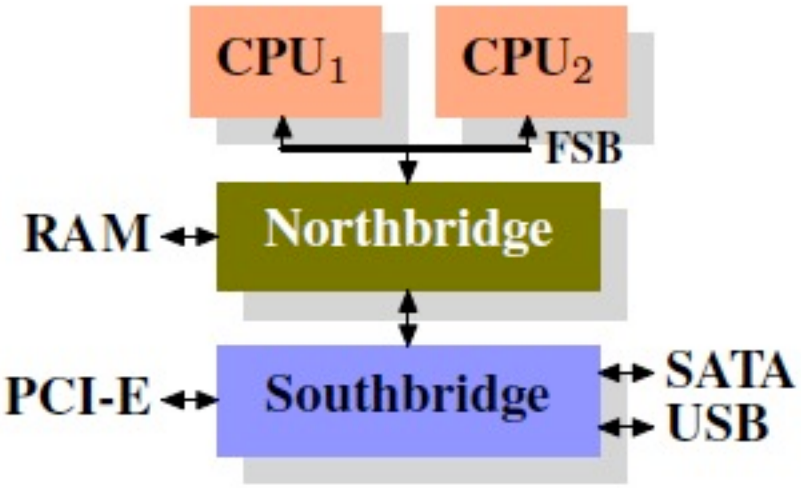
NUMA + memory affinity

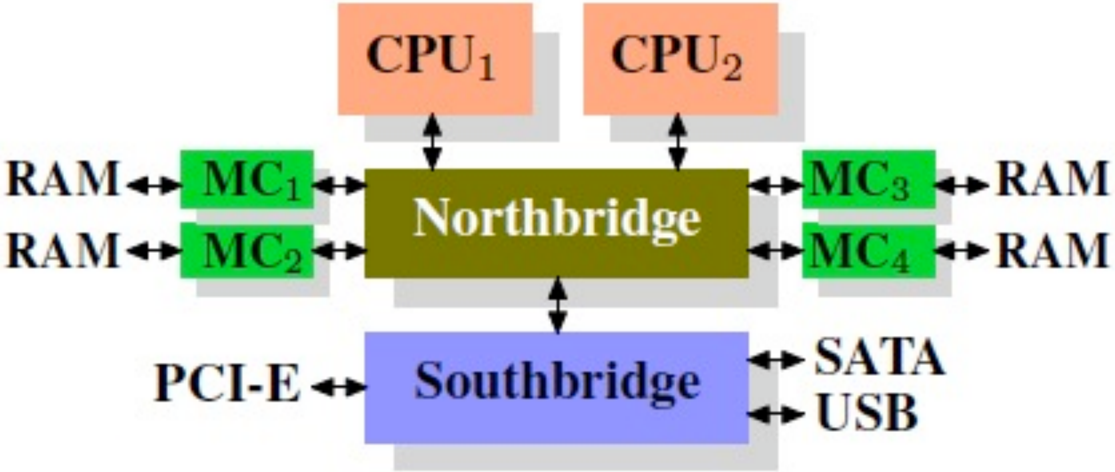
| -lsize=12 | MPI | LWP | DRAM |
|---|---|---|---|
| aprun -n <1-12> | 1 - 12 | 1 | 2 * 2^30 |
| aprun -n 2 -sn 2 -S 1 -d 6 | 2 | 1 - 6 | 12 * 2^30 |
| aprun -n 1 -N 1 -d 16 | 1 | 1 - 12 | 24 * 2^30 |

<-S> * <-d> cannot exceed the maximum number of CPUs per NUMA node

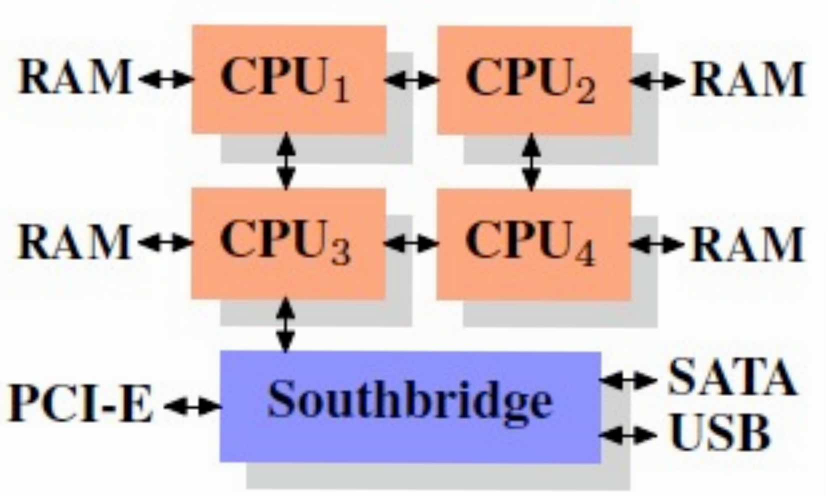


no NUMA, 6 PEs/socket





external memory control



cpu integrated memory control

# Concurrency is when processes or threads share hardware resources

## CPU

- **ALU**, adds, comparisons
- **FPU**, floating point operations
- **L/S U**, data, ins loads / stores
- **Registers**, fast memory; FPR, GPR, etc.
- **PC**, program counter -address in memory of instruction that is executing (control flow, fetch / decode in CPU)
- **Memory interface**, often L1 and L2 caches

**other:**

clock speed
buses
ISA (Intel x86 most popular, x86-64, ...)

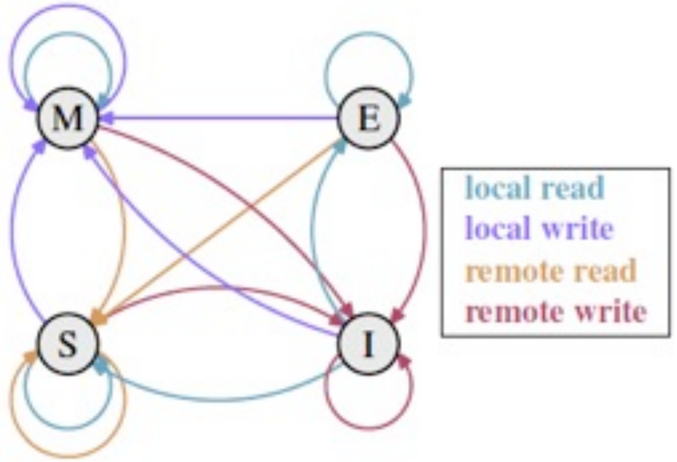
improves performance for local data refs.
-still forced to communicate / orchestrate for non-local

# Use of threads means coping with complicated issues

**threaded**

- cache contention, coherency
- atomicity
- memory bandwidth
- scheduling, pinning to hardware

**need tools**
**identification of relevant observables**



***Modified***, local processor has only copy of data and modifies it

***Exclusive***, CL is not modified and not in another core's cache

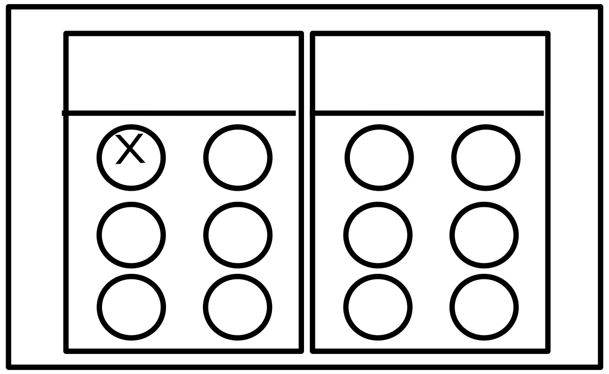
***Shared***, CL not modified -might be in cache somewhere

***Invalid***, CL is invalid -not used

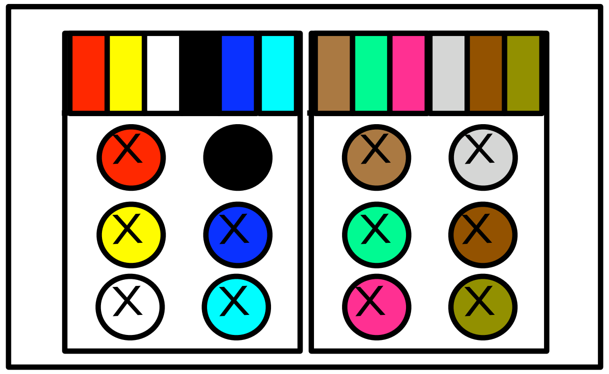
*other processor's activities are snooped on the address bus

fork (create) / join overheads

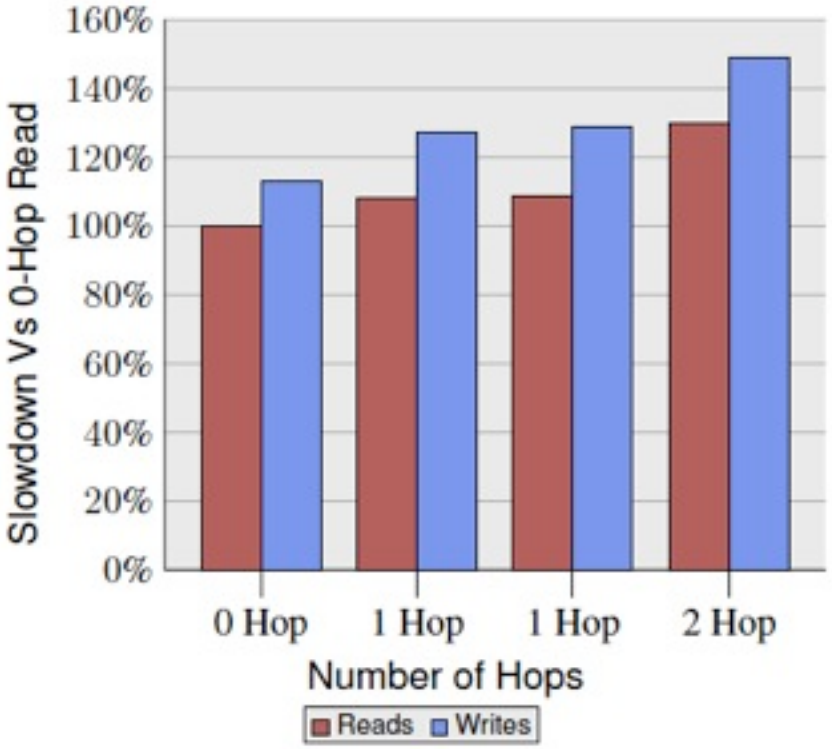
| NT | Cycles | L2DCM |
|---|---|---|
| 1 | 1959379 | 69 |
| 2 | 2020818 | 81 |
| 4 | 2289393 | 122 |
| 6 | 2366367 | 146 |
| 8 | 2499159 | 239 |



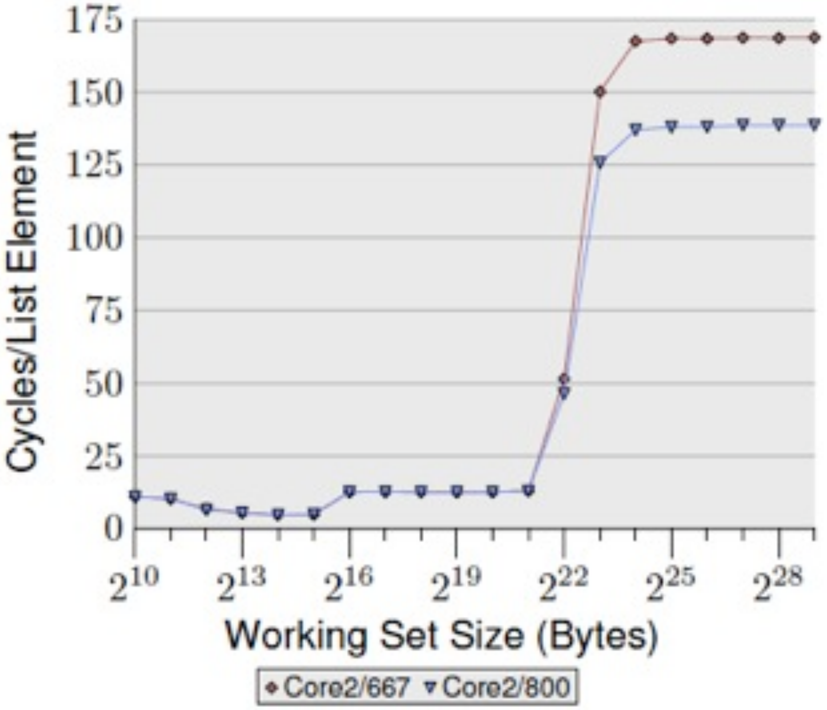
NUMA + memory affinity

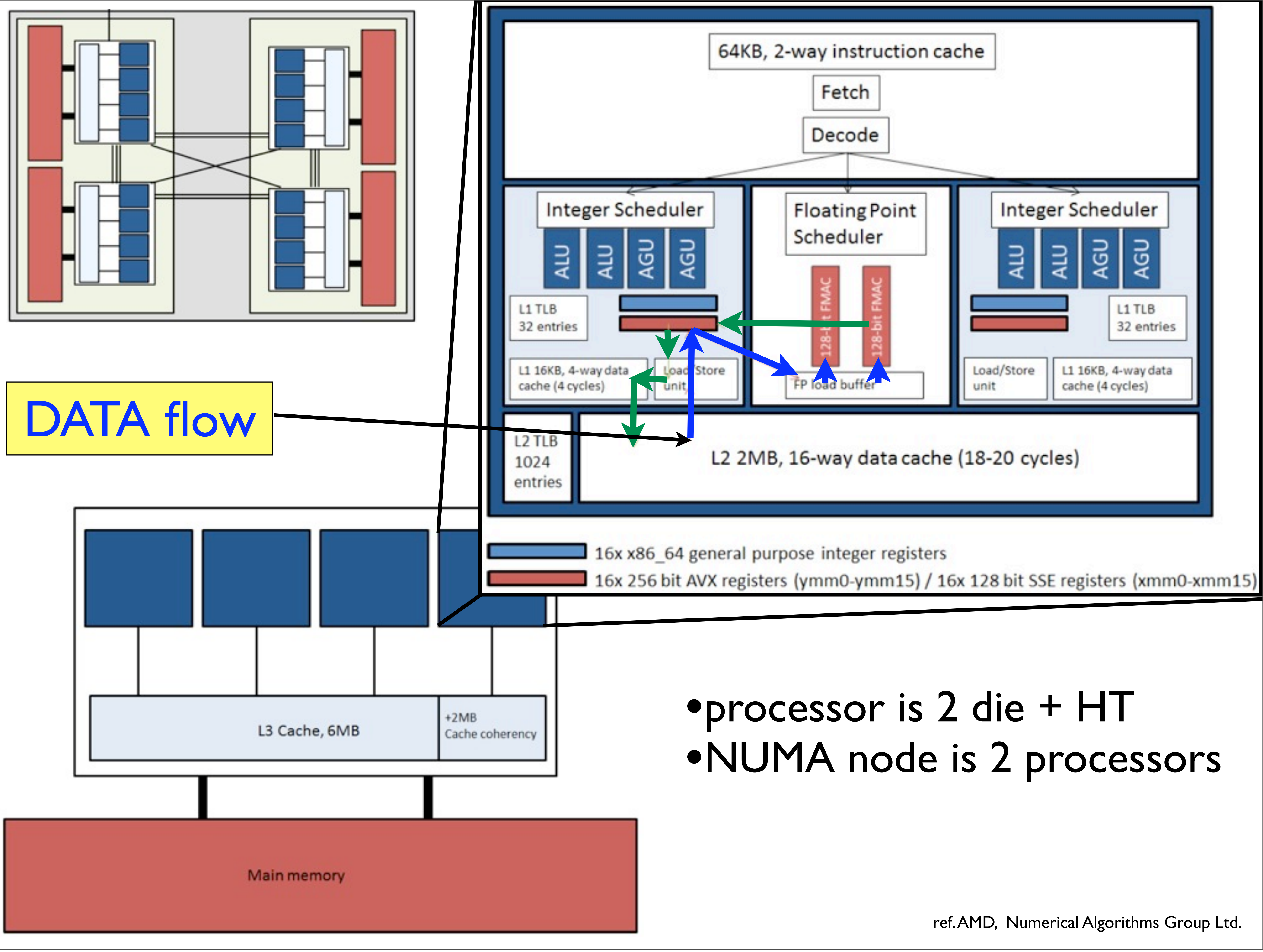


no NUMA, 6 PEs/socket



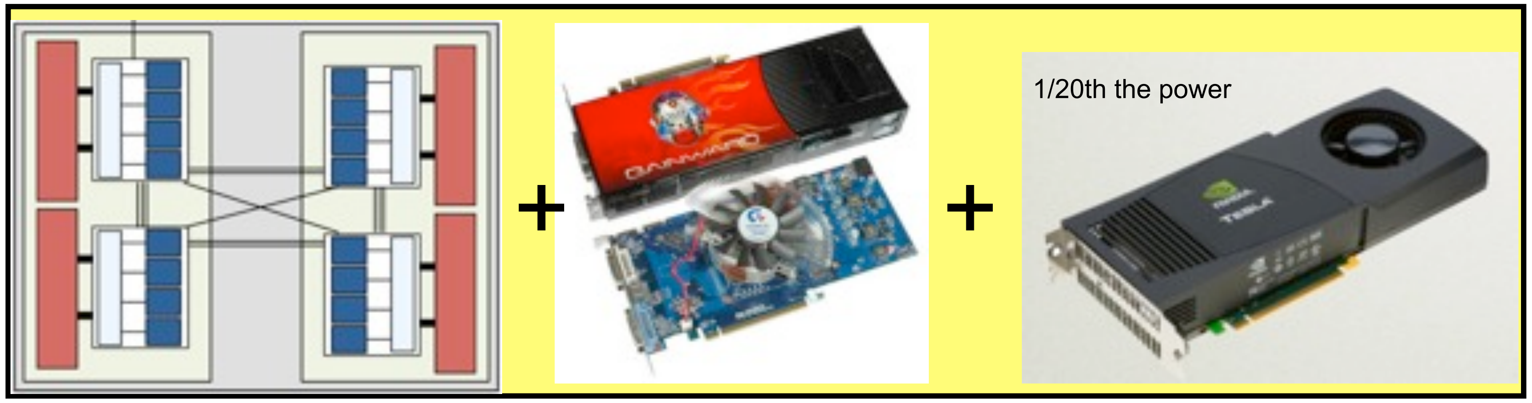
make the FSB faster with increasing core count



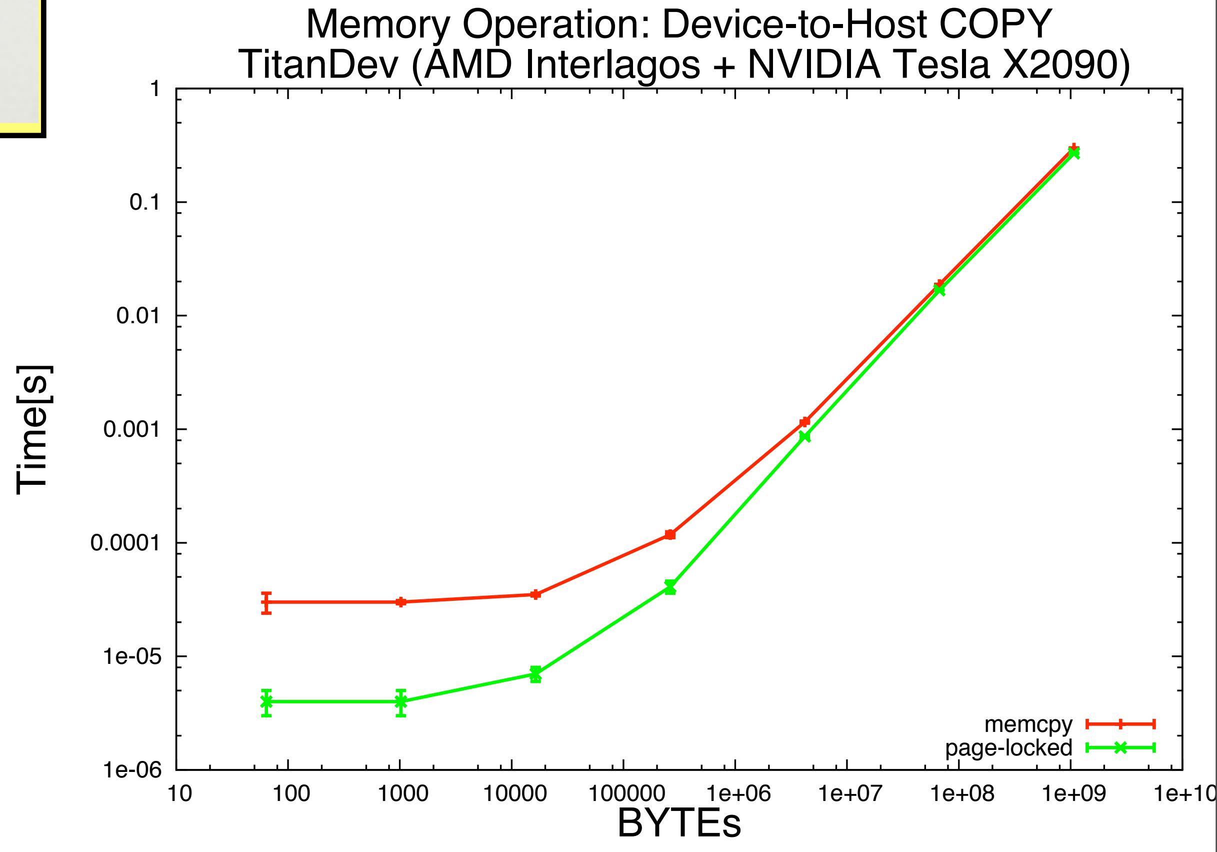

DATA flow
64KB, 2-way instruction cache
Fetch
Decode
Integer Scheduler
ALU
ALU
AGU
AGU
Floating Point Scheduler
128-bit FMAC
128-bit FMAC
Integer Scheduler
ALU
ALU
AGU
AGU
L1 TLB 32 entries
L1 16KB, 4-way data cache (4 cycles)
Load/Store unit
FP load buffer
Load/Store unit
L1 16KB, 4-way data cache (4 cycles)
L1 TLB 32 entries
L2 TLB 1024 entries
L2 2MB, 16-way data cache (18-20 cycles)
🟦 16x x86_64 general purpose integer registers
🟥 16x 256 bit AVX registers (ymm0-ymm15) / 16x 128 bit SSE registers (xmm0-xmm15)
L3 Cache, 6MB
+2MB Cache coherency
Main memory

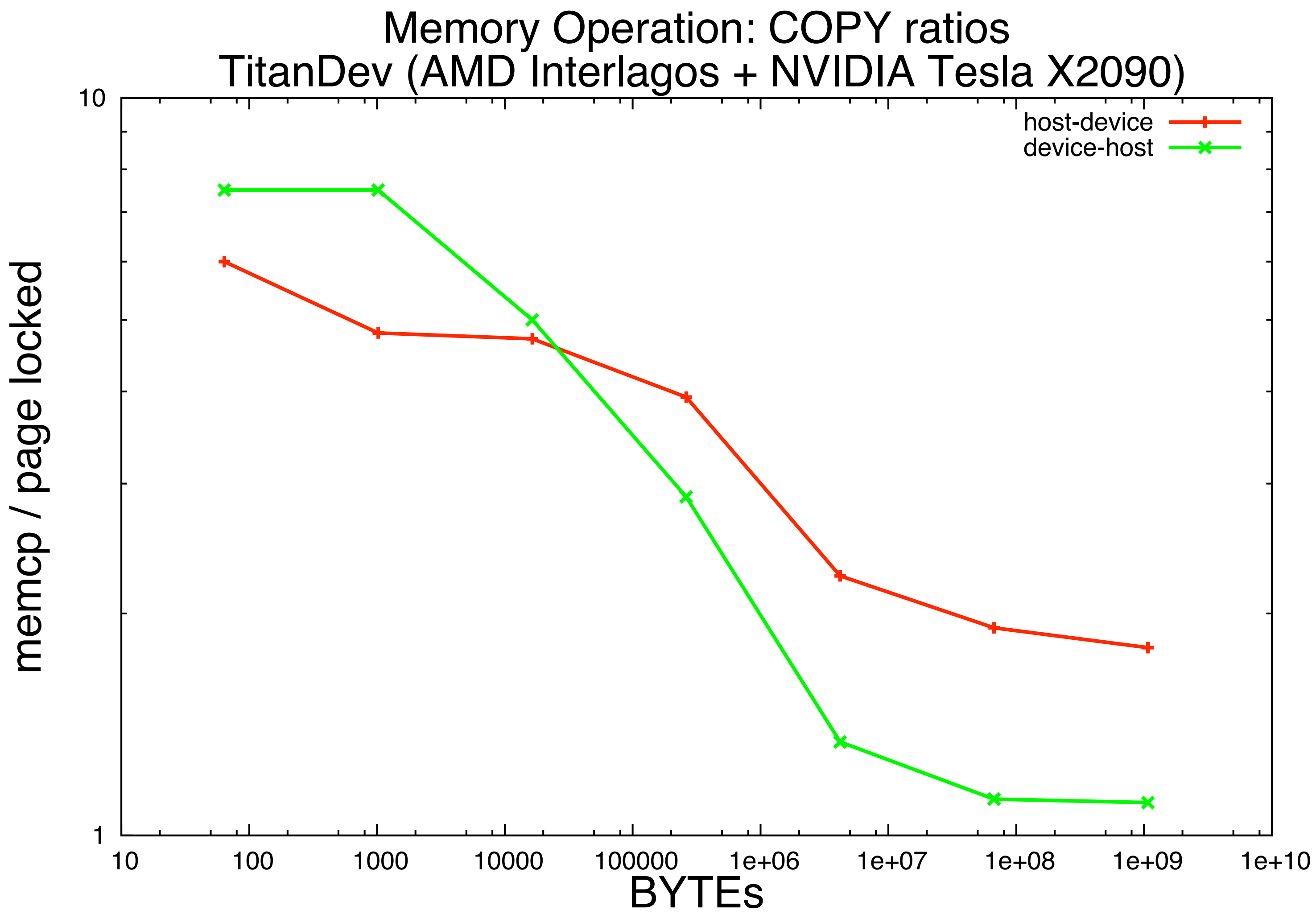
- processor is 2 die + HT
- NUMA node is 2 processors

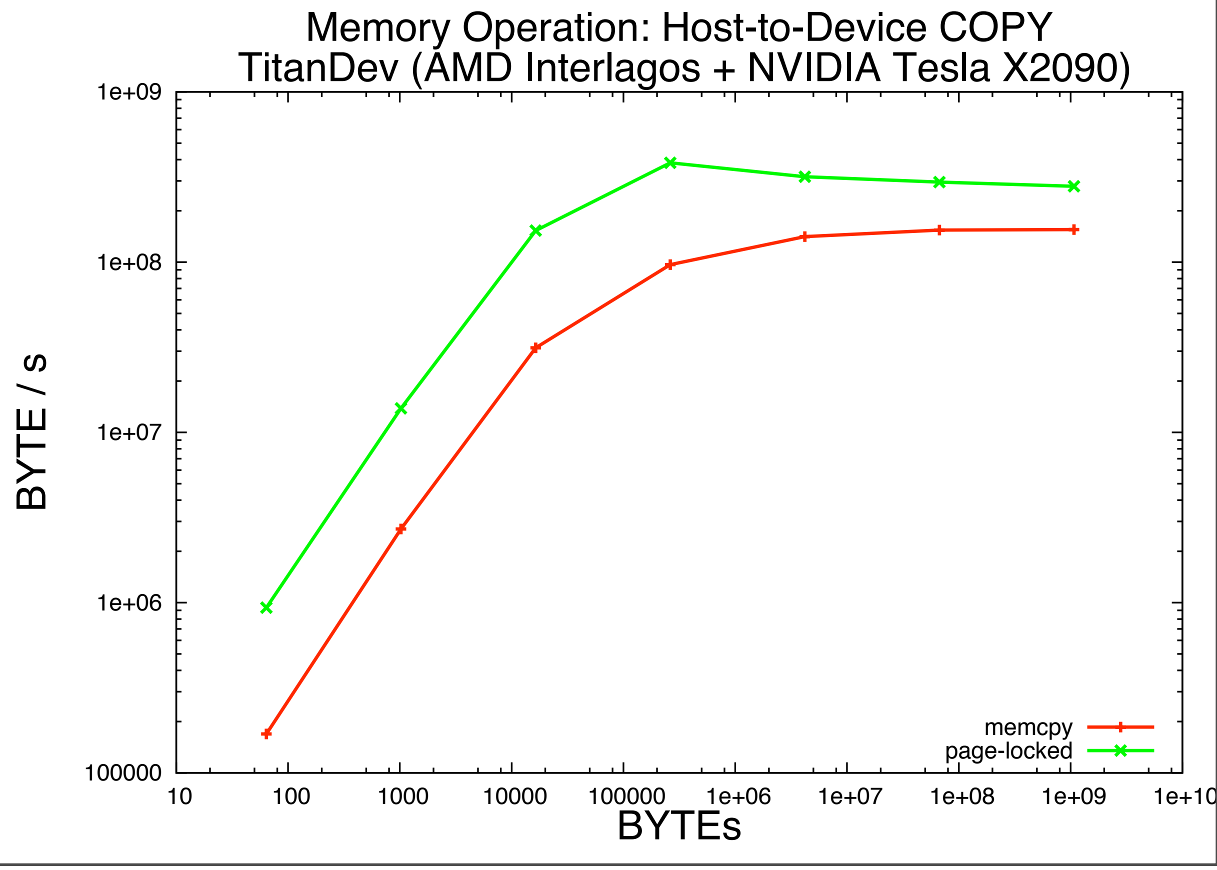
ref. AMD, Numerical Algorithms Group Ltd.



- deeper memory, heterogeneous hardware, distinct binaries

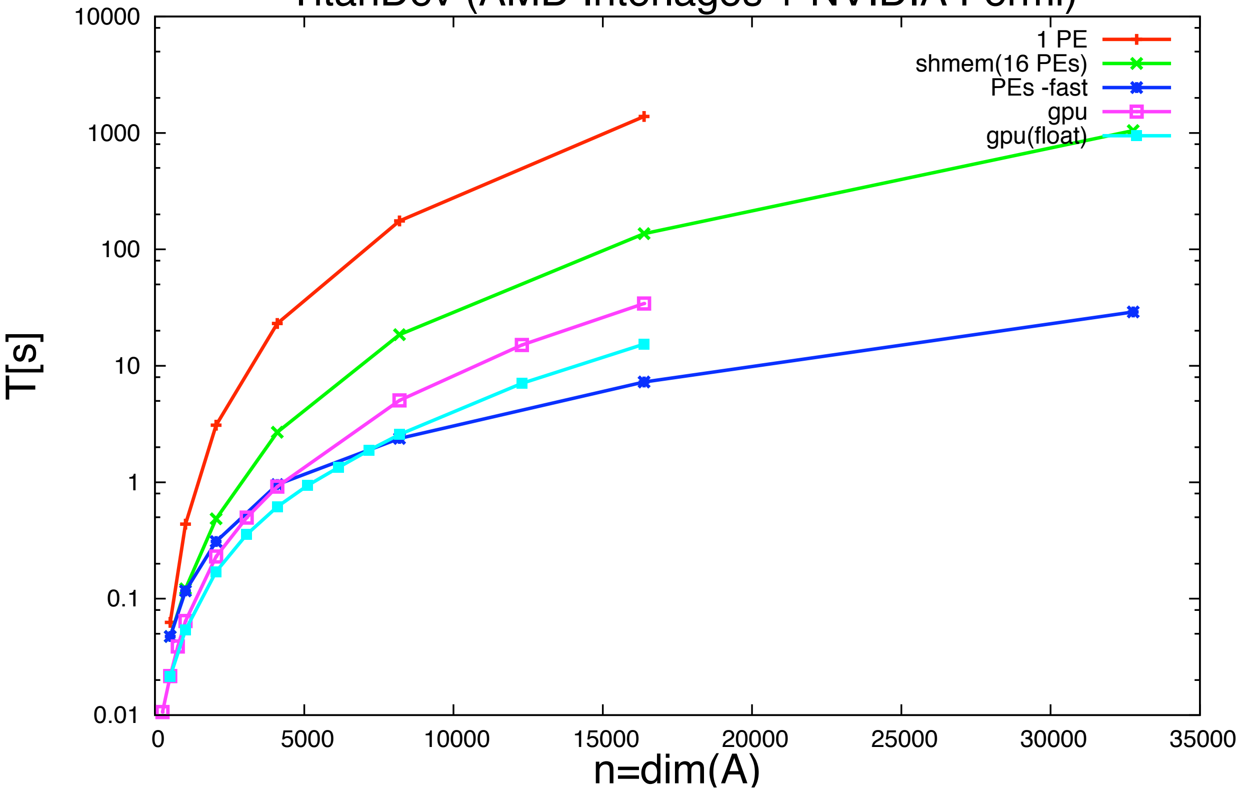




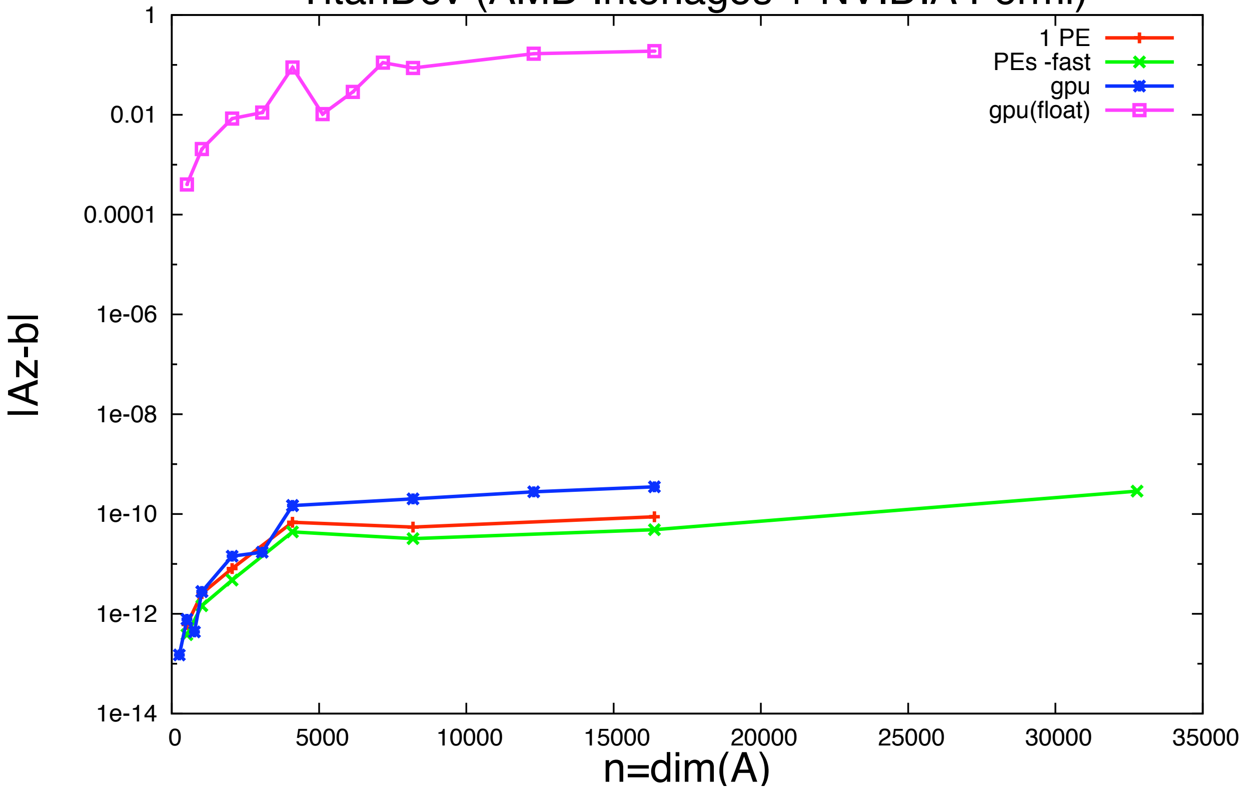


# Other Basic Examples:

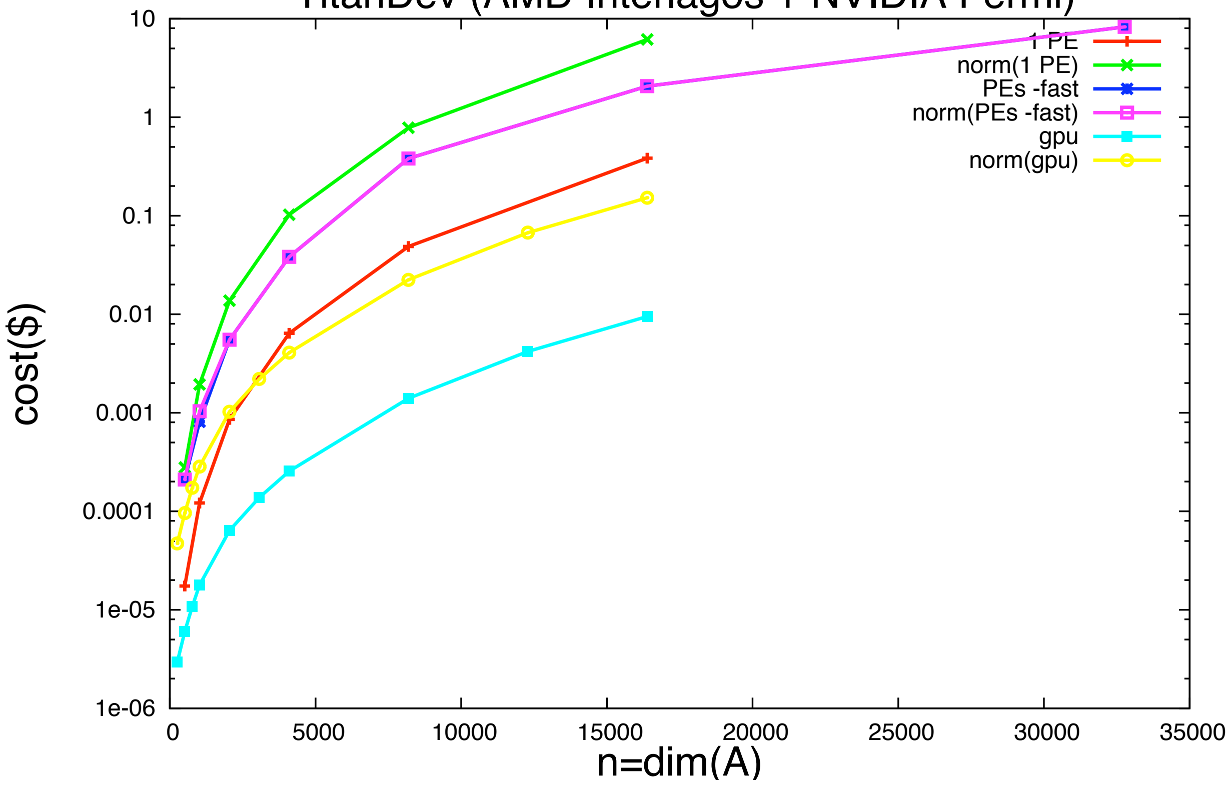
Az=b Timings
TitanDev (AMD Interlagos + NVIDIA Fermi)



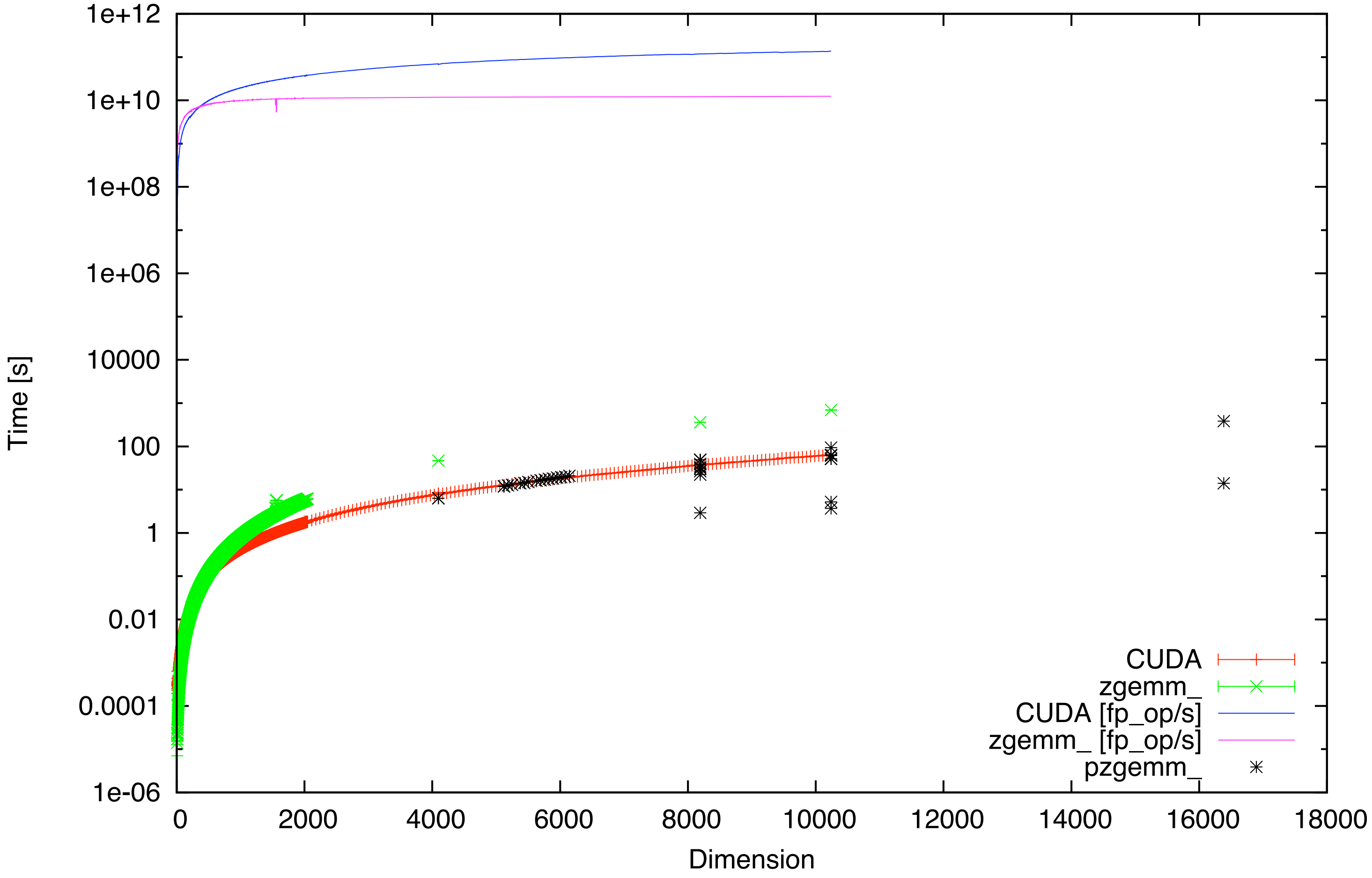
Numerical Error |b-Az| / |A|*|z|*eps
TitanDev (AMD Interlagos + NVIDIA Fermi)



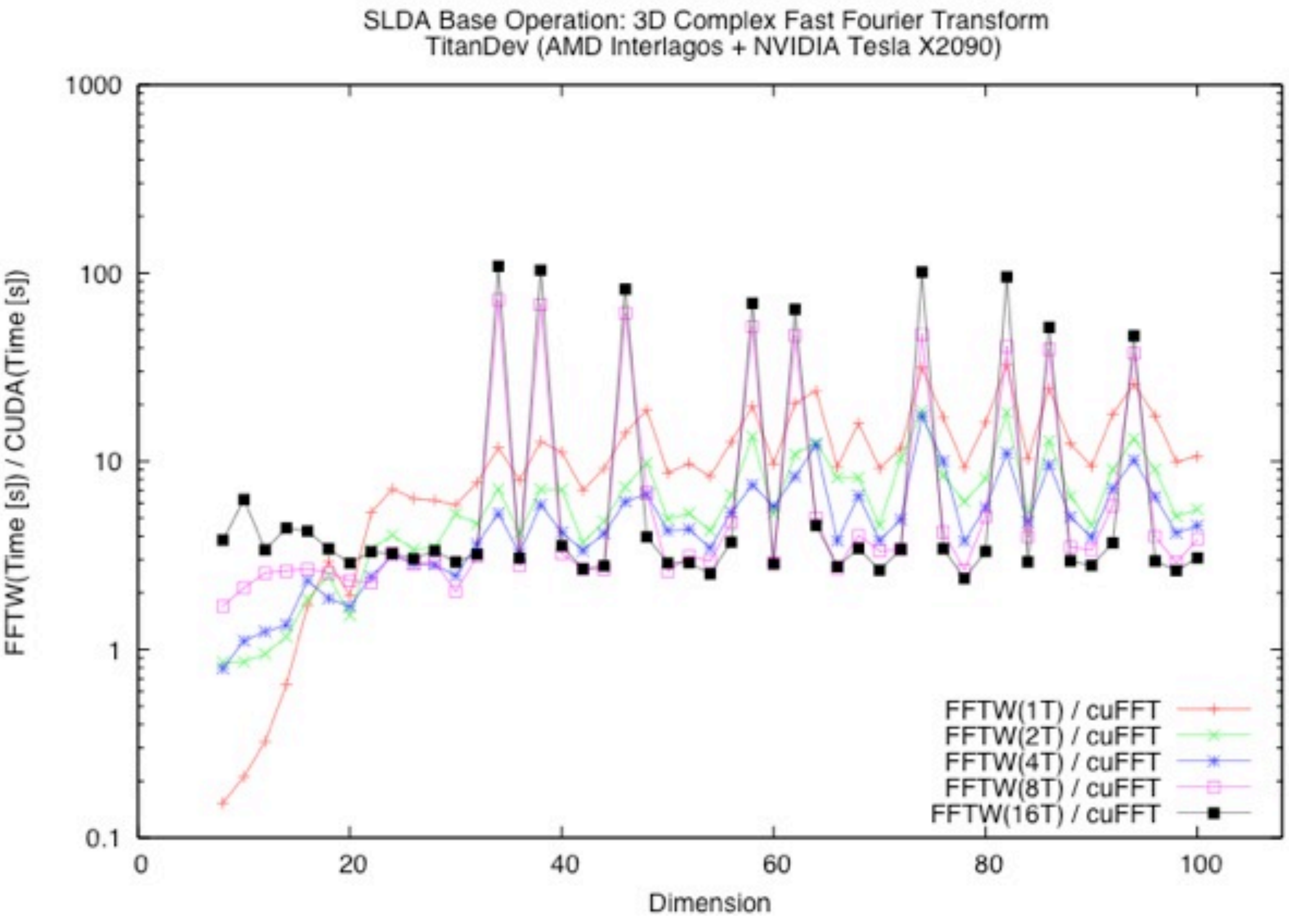
Cost(Az=b)
TitanDev (AMD Interlagos + NVIDIA Fermi)



CUDA BLAS (cublas) Square Matrix Multiplication
TitanDev (AMD Interlagos + NVIDIA Fermi)



# Multi-threaded versus GPU



i.e. apply operators *F(derivatives)* to plane wave based functions

**Pi** = 3.1415926535 8979323846 2643383279…

```
[rochekj@clue-1]$ time ./xpt_pi 1
1 threads:
        pi = 3.14159  (3.14159265358970752)
       walltime:= 12.2492 [s]
       cpu:= 12.24 [s]
12.247u 0.000s 0:12.25 99.9%    0+0k 0+0io 0pf+0w
[rochekj@clue-1]$ time ./xpt_pi 2
2 threads:
        pi = 3.14159  (3.14159265359007167)
       walltime:= 6.16762 [s]
       cpu:= 12.27 [s]
12.272u 0.000s 0:06.16 199.1%   0+0k 0+0io 0pf+0w
[rochekj@clue-1]$ time ./xpt_pi 3
3 threads:
        pi = 3.14159  (3.14159265358914352)
       walltime:= 4.13565 [s]
       cpu:= 12.24 [s]
12.249u 0.000s 0:04.13 296.3%   0+0k 0+0io 0pf+0w
[rochekj@clue-1]$ time ./xpt_pi 4
4 threads:
        pi = 3.14159  (3.14159265358976824 7)
       walltime:= 3.09586 [s]
       cpu:= 12.22 [s]
12.226u 0.000s 0:03.09 395.4%   0+0k 0+0io 0pf+0w
[rochekj@clue-1]$ time ./xpt_pi 45
45 threads:
        pi = 3.14159  (3.14159265358979 0896)
       walltime:= 3.13209 [s]
       cpu:= 12.23 [s]
12.230u 0.000s 0:03.13 390.7%   0+0k 0+0io 0pf+0w
```

bonus, from R. Feynman

## Computing in the Future

- programming too complex today
  - smarter machines -better human machine interactions
  - there are too many programming languages
    - choose one and make it standard

- debugging means to fix errors in a program or a machine
  - need automatic debugging

bonus, from R. Feynman

# Computing in the Future

- make physical components in 3D not simply constrained to surface of a chip

- a device to detect defective elements
  - automatically rewire to avoid defective elements

bonus, from R. Feynman

# Computing in the Future

parallel computers

energy consumption of machines

- could be made less with time varying voltage
- go slower but use less energy, allows for increases in hardware units

physical limits of physical devices

- reversible logic circuitry, Bennett, Landauer, Scientific American
    - transistor can go forward and backward
        - can recover input from output